MFIRE-2: A MULTI AGENT SYSTEM FOR FLOW-BASED INTRUSION DETECTION USING STOCHASTIC SEARCH

THESIS

Timothy J. Wilson, Captain, USAF
AFIT/GCO/ENG/12-12

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCO/ENG/12-12

MFIRE-2: A MULTI AGENT SYSTEM FOR FLOW-BASED INTRUSION DETECTION USING STOCHASTIC SEARCH

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science

Timothy J. Wilson, BCEN

Captain, USAF

March 2012

AFIT/GCO/ENG/12-12

MFIRE-2: A MULTI AGENT SYSTEM FOR FLOW-BASED INTRUSION DETECTION USING STOCHASTIC SEARCH

Timothy J. Wilson, BCEN
Captain, USAF

Approved:

Dr. Gary B. Lamont (Chairman)	Date
Dr. Barry E. Mullins (Member)	Date
Dr. Gilbert L. Peterson (Member)	Date

Abstract

Detecting attacks targeted against military and commercial computer networks is a crucial element in the domain of cyber warfare. Intrusions, Denial of Service attacks and Worm propogations are an ever present threat, and defending networks from hostile action has become a top priority for both policy makers and network administrators. The traditional method of signature-based intrusion detection is a primary mechanism to alert administrators to malicious activity. However, signature-based methods are not capable of detecting new or novel attacks. In addition, increasing networking line speeds is making it more difficult to monitor packets and detect malicious signatures. Thus, signature-based ID is relegated to monitoring a small sample of the total traffic, increasing the likelihood of malicious traffic entering the network system without scrutiny.

Further characterizing traditional ID is the location from which it is performed, resulting in network-based and host-based ID systems. Network-based ID has a broad perspective enabling detection of attacks that are distributed in nature, but may not protect individual systems effectively without incurring large bandwidth penalties while collecting data from all hosts. Host-based ID has a comprehensive view of local systems, but may not be able to detect distributed malicious activity effectively. A *multi agent* design paradigm leverages the strengths of both *network*-based and *host*-based ID methods.

While deep packet inspection is only possible on a small subset of messages, flow-based metrics can be applied universally. Comprehensive analysis of a large percentage of network traffic is possible in real time, if a higher level of observation is used. This method becomes extremely effective when analyzed from multiple points in the

network. *flow*-based intrusion detection complements traditional *signature*-based ID systems.

This research continues development of a novel simulated, multiagent, flow-based intrusion detection system called MFIRE. Agents in the network are trained to recognize common attacks, and share data with other agents to improve the overall effectiveness of the system. A Support Vector Machine (SVM) is the primary classifier with which agents determine if an attack is occurring. Agents are prompted to move to different locations within the network to find better vantage points, and two methods for achieving this are developed. One uses a centralized reputation-based model, and the other uses a decentralized model optimized with stochastic search. The latter is tested for basic functionality. The reputation model is extensively tested in two configurations and results show that it is significantly superior to a system with non-moving agents. The resulting system, MFIRE-2, demonstrates exciting new network defense capabilities, and should be considered for implementation in future cyber warfare applications.

Acknowledgements

I would like to thank all who gave me the inspiration to make this thesis a reality.

My classmates gave continual support throughout our time at AFIT, providing a sounding board for many ideas and improvements. But it was the time away from work that gave me the most assistance. I thank the many friends I have developed over our 18 month endeavor for sharing in both good and tough times, and for their true Wingman spirit.

And especially I thank Dr. Lamont for his continued drive and encouragement. Without it I would not have been able to compile such a product as this. I have learned so much from his efforts, and very few individuals have shown such dedication to my success in the Air Force.

Timothy J. Wilson

Table of Contents

Page

List of Figures

Figure Page

List of Tables

List of Abbreviations

Abbreviation		Page
IDS	Intrusion Detection System	5
MAS	Multi Agent Systems	5
ID	Intrusion Detection	5
BGP	Border Gateway Protocol	9
CD	Controlled Distance	9
FKP	Fabrikant-Koutsoupias-Papadimitriou	10
HIDS	Host-based Intrusion Detection System	12
NIDS	Network Intrusion Detection System	12
UDP	User Datagram Protocol	17
ICMP	Internet Control Message Protocol	17
SVM	Support Vector Machines	26
MAS	Multiagent System	31
EA	Evolutionary Algorithm	36
GAs	Genetic Algorithms	36
ESs	Evolution Strategies	36
EP	Evolutionary Programming	36
RPC	Remote Procedure Calls	46
TCP	Transmission Control Protocol	49
UDP	User Datagram Protocol	49
IANA	Internet Assigned Numbers Authority	49
HTTP	Hyper Text Transfer Protocol	49
MOGAs	Multi-Objective Genetic Algorithms	94

Abbreviation Page

MOMA Multi Objective Memetic Algorithm 94

PD projection distance 94

MFIRE-2: A MULTI AGENT SYSTEM FOR FLOW-BASED INTRUSION DETECTION USING STOCHASTIC SEARCH

I. Introduction

The Department of Defense is beginning a massive reduction in spending over the next decade [11]. In spite of this, investment in cyberspace capabilities continues to grow [11]. Thus, the DoD has outlined a strategy to invest in new capabilities to maintain a decisive edge in all aspects of cyber:

> *Modern armed forces cannot conduct high-temp, effective operations without reliable information and communication networks and assured access to cyberspace and space. Today space systems and their supporting infrastructure face a range of threats that may degrade, disrupt or destroy assets. Accordingly, DoD will continue to work with domestic and international allies and partners and invest in advanced capabilities to defend its networks, operational capability and resiliency in cyberspace and space [11].*

Strategic Initiative 2 of the DoD Strategy for Operating in Cyberspace is to "employ new defense operating concepts to protect DoD networks and systems [2]." This document highlights that *a primary emphasis must be placed on protecting military networks*. As the DoD becomes more reliant on networking as a way to conduct operations, so too do the stakes become higher. Malicious users and other powers continually push the boundaries of innovation to discover new vulnerabilities [27, 90, 30]. Intrusion detection in particular is a crucial element in this fight. The ability to detect an active attempt to exploit a network is paramount. After all, no reaction can be attempted if one does not know they are under attack in the first place.

1.1 Network Threats

Three major classes of network threats include [84]:

- Attacks that consume network resources, denying their use for legitimate purposes
- Attacks that infiltrate systems, allowing attackers unauthorized access to system resources, including sensitive data, data storage, privileged relationships with other systems, and network connectivity
- Unauthorized vulnerability scans, providing attackers vital reconnaissance in preparation for infiltrating activities

These three threats are mutually reinforcing. For example, a successful scan allows an attacker to infiltrate networks with great stealth and precision; once in control of multiple hosts, the attacker may use them to launch a distributed denial of service attack on another target system or network. Alternatively, the attacker can use these newly acquired assets to conduct further scans more efficiently / stealthily. As another example, a clever attacker may launch a denial of service attack on a highly visible service to divert the attention of security personnel from his infiltration activities.

Within these categories, many types of intrusion are recognized [62]:

Information Gathering—Network devices can be discovered and profiled in much the same way as other types of systems. Attackers usually start with port scanning. After they identify open ports, they use banner grabbing and enumeration to detect device types and to determine operating system and application versions. Armed with this information, an attacker can attack known vulnerabilities that may not be updated with security patches.

Sniffing—Sniffing or eavesdropping is the act of monitoring traffic on the network for data such as plaintext passwords or configuration information. With a simple

packet sniffer, an attacker can easily read all plaintext traffic. Also, attackers can crack packets encrypted by lightweight hashing algorithms.

Spoofing—Spoofing is a means to hide one's true identity on the network. To create a spoofed identity, an attacker uses a fake source address that does not represent the actual address of the packet. Spoofing may be used to hide the original source of an attack or to work around network access control lists (ACLs) that are in place to limit host access based on source address rules.

Session Hijacking—Also known as man in the middle attacks, session hijacking deceives a server or a client into accepting the upstream host as the actual legitimate host. Instead the upstream host is an attacker's host that is manipulating the network so the attacker's host appears to be the desired destination.

Denial of Service—Denial of service denies legitimate users access to a server or services. The SYN flood attack is a common example of a network level denial of service attack. It is easy to launch and difficult to track. The aim of the attack is to send more requests to a server than it can handle. The attack exploits a potential vulnerability in the TCP/IP connection establishment mechanism and floods the server's pending connection queue.

Viruses, Trojan Horses, and Worms—A virus is a program that is designed to perform malicious acts and cause disruption to the operating system or applications. A Trojan horse resembles a virus except that the malicious code is contained inside what appears to be a harmless data file or executable program. A worm is similar to a Trojan horse except that it self-replicates from one server to another. Worms are difficult to detect because they do not regularly create files that can be seen. They are often noticed only when they begin to consume system resources because the system slows down or the execution of other programs halt. The Code Red Worm is one of the most notorious to afflict IIS; it relied upon a buffer overflow vulnerability

in a particular ISAPI filter. The success of these attacks on any system is possible through many vulnerabilities such as weak defaults, software bugs, user error, and inherent vulnerabilities in Internet protocols.

Footprinting—Examples of footprinting are port scans, ping sweeps, and NetBIOS enumeration that can be used by attackers to glean valuable system-level information to help prepare for more significant attacks. The type of information potentially revealed by footprinting includes account details, operating system and other software versions, server names, and database schema details.

Password Cracking—If the attacker cannot establish an anonymous connection with the server, he or she will try to establish an authenticated connection. For this, the attacker must know a valid username and password combination. Unchanged default account names, and the use of blank or weak passwords makes the attacker's job even easier.

Arbitrary Code Execution—If an attacker can execute malicious code on the server, the attacker can either compromise server resources or mount further attacks against downstream systems. The risks posed by arbitrary code execution increase if the server process under which the attacker's code runs is over-privileged. Common vulnerabilities include weak IIS configuration and unpatched servers that allow path traversal and buffer overflow attacks, both of which can lead to arbitrary code execution.

Unauthorized Access—Inadequate access controls could allow an unauthorized user to access restricted information or perform restricted operations. Common vulnerabilities include weak IIS Web access controls, including Web permissions and weak NTFS permissions.

1.2 Multiagent Intrusion Detection

Intrusion Detection Systems are often employed as gatekeepers for a local area network. The principle focus of the system, whether host-based or network-based, is to monitor local traffic for signs of malicious activity. Little or no effort is expended to *share* the obtained information with another Intrusion Detection System (IDS). Indeed, there is potential gain in compiling information over a much broader view of the network. IDSs in different autonomous systems that share information with each other would be much better placed to develop a collective view of threats progressing accross the networks.

Taken from another viewpoint, if threats are unpredictable in their location, would it not be beneficial to move the observing IDS *to* that point, rather than wait for the attacker to strike on their own terms? To perform this task, we explore the concept of a multi-agent IDS, in which individual agents are able to move throughout a network, and share data to collectively determine if an attack is occurring.

Multi Agent Systems (MAS) for Intrusion Detection (ID) is not a new concept. Herrero [40] summarizes several multi-agent IDSs, and lists some key areas where MAS may be appropriate:

- The environment is open, highly dynamic, uncertain, or complex
- Agents are a natural metaphor—Many environments are naturally modeled as societies of agents, either cooperating with each other to solve complex problems, or else competing with one-another.
- Distribution of data, control or expertise—A centralized solution is at best extremely difficult or at worst impossible.
- Legacy systems—Technologically obsolete software but functionally essential to an organization. One solution to this problem is to wrap the legacy components,

providing them with an "agent layer" functionality.

1.3 Goal and Objectives

The *goal* of this research is to continue the development of a scalable software architecture for a multi-agent, flow-based intrusion detection system. The following high-level *objectives* support this goal:

- Design and evaluate a multi-agent intrusion detection system using a Reputation system
- Design and evaluate a multi-agent intrusion detection system using stochastic search

The effectiveness of these two models is compared to the baseline stationary model. Research results include an evaluation of the environment's classification performance. The main output of this research is an effective and efficient simulation environment to conduct ongoing, flow-based intrusion defense experiments.

1.4 Approach

This research introduces a framework for conducting simulations of networks under attack, and a multiagent system which is trained to detect threats. Individual agents can be trained using various techniques, and tested in the simulator to validate their effectiveness. The strength of the environment is its open object-oriented nature, which allows agents and processes of any type to be instantiated. By keeping standardized network and traffic models, one can test a wide variety of agents under varying attack models, and qualitatively compare their performance.

A central feature of this framework is the movement of multiple agents within the network, which allows the agents to find better vantage points for classifying an attack.

For this iteration of the design, we seek to compare the effectiveness of two distinct models for attack identification. These models use the same underlying classifier, but vary in the way that agents' movement decisions are made. A reputation system is used to allow a central controller to dictate agents' movement decisions. For a comparison, we also examine the case when agents are at a fixed random location and no movement is allowed. Finally, the agents are allowed to move freely on their own, with their behavior optimized using a genetic algorithm. Throughout this document, these are known as the *reputation model*, and *free-movement model*, respectively.

This research follows from two previous efforts by Eric Holloway [41] and David Hancock [37]. In particular, we seek to continue work by David Hancock which tests the hypothesis that a flow-based, multi-agent network attack classifier can be made more effective by employing a reputation system to govern agent mobility. We also seek to include work inspired by Eric Holloway that created a self-organized multi-agent network security system using Evolutionary Computation. The two environments created for these theses used fundamentally different approaches, and because of this comparing results between the two is difficult. By taking the best elements of both, a single integrated simulation environment can be created which allows a wide variety of experiments to be run regarding multi-agent flow-based intrusion detection. This research includes executing initial baseline experiments to test our combined approach, and we demonstrate the potential usefulness of such an environment.

1.5 Thesis Overview

This chapter frames the problem and introduces an approach to solving it. Chapter II explores the concepts involved in realizing this approach, including threat modeling, flow-based ID, and evolutionary computation. Chapter III details the design of our multi agent system called MFIRE-2. Chapter IV presents the experimental per-

formance analysis of MFIRE-2. Chapter V concludes with a summary of the research impact and opportunities for future research.

Results from two experiments demonstrate that MFIRE's Reputation system allows the agents to find vantage points in the network that increase the system's classification accuracy. In addition, the process of using an alternate model for agent movement is explored. Stochastic search provides a method for agents to optimize a local movement actuator, so that movement is decentralized, and no longer tied to a central Reputation scheme.

An important contribution of this thesis is the developed network simulation environment, MFIRE-2, which provides a scalable, object-oriented framework to execute network threat analysis using robust multiagent, flow-based techniques. This environment supports not only the investigation of the subject multi agent system, but may be used for other network research investigations as well where the level of abstraction is suitable for the purpose.

II. Background

This investigation focuses on a subset of network-based attacks. Specifically, it focuses on the challenge of recognizing that a flow-based attack is taking place. This chapter prepares for that challenge by discussing first the concepts and current research in critical areas relevant to the flow-based attack classification system.

2.1 Network Topology and Routing

The Internet is a network of nodes, comprised of sub-networks. The collection of routers and other networked devices under the same administrative control is called an *Autonomous System* (AS), and *gateway routers* are responsible for forwarding traffic to and from other autonomous systems. Although each AS may handle traffic internally in unique ways, all rely on a common backbone networking protocol for inter-autonomous system routing called Border Gateway Protocol (BGP). See [54] for an overview of autonomous systems and BGP.

Two primary modeling concepts within this domain are *topology* and *traffic*. With a firm grasp of these principle aspects, we can devise a system that achieves a good balance between efficiency and accuracy. Simulation provides a safe and robust environment to test our approach and gain greater confidence in its defense characteristics.

As yet, there is no golden bullet for internet topology modeling. However, Spatharis et al. present a well-balanced approach called the Controlled Distance (CD) Model [82]. This model balances two key aspects topology: commonly used power-law models, as found in [26]; and a general approach to "rely on domain knowledge and exploit the details that matter when dealing with a highly engineered system such as the Internet" [92].

CD is an updated treatment of an earlier model called *Fabrikant-Koutsoupias-*

Papadimitriou (FKP), and the goal is to address the need for edges between nodes that are not quite leaves, nor particularly central, but are of intermediate centrality. As each node i is added to the network and linked to the node j, a second edge is attached from j to another node k minimizing

$$min_k\{\alpha \cdot d_{jk} + ecc(k)\} \tag{1}$$

over all k such that the hop distance from j to k is at most a constant c. In this equation, d_{ij} is the Euclidean distance between the nodes and represents the "last mile cost." The relative importance of this objective is controlled via the weight α. The second term is the *eccentricity* of j and captures the distance from j to the center.

This model decreases the power law exponent while having high average degree and several leaves. The authors of [82] declare this to be, in many ways, the "best performing" of their models in achieving similarity to the Internet's AS graph. This model and various alternatives are packaged by the authors in the package TopGen.

Other topology generators include:

- Tiers [24]
- GT-ITM - Georgia Tech Internetwork Topology Models [14]
- Inet [94]
- nem [57]
- BRITE [61]
- GDTANG - Geographic Directed Tel Aviv University Network Generator [8]
- RealNet [19], [18]

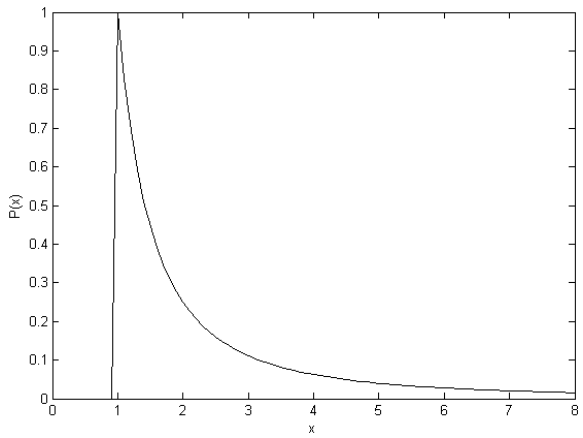


Figure 1. Probability density function for Pareto distribution, $\alpha = 1.0$**,** $b = 1.0$

RealNet relies on publicly available datasets including BGP tables and traceroute records, as does [28], but addresses some of the problems inherent in these datasets and does not attempt to fit specific power-law-based statistics. For example, it gives direct consideration to the IP-aliasing problem, whereby more routers may be inferred than actually exist because each router has a different IP address for each of its interfaces. It also factors in likely policy relationships between neighboring autonomous systems [37]. RealNet is promising but not available for the current research. In the meantime, FKP/CD provides a reasonable approach.

In addition to modeling network topology, simulated internet traffic must also be examined. A widely used model for packet routing is the Poisson model. Willinger and Paxson [93] advocate against the Poisson model for better, more fractal-like traffic distribution models [37]. For network traffic, a Pareto model may be preferred.

The Pareto model exhibits scale-invariant behavior [33]. It has a density function $P(X) = \frac{\alpha b^\alpha}{x^{\alpha+1}}$, $x \geq b$, which has a heavy tail [52]. Figure 1 shows an example for $\alpha = 1.0$, $b = 1.0$. Willinger and Paxson explain that this heavy tail accounts for the fractal nature of aggregated network traffic [93]. To generate a random Pareto-distributed sample, inverse transform sampling is used. Given a random variable U drawn from the uniform distribution $(0, 1)$, T, is Pareto-distributed [23], and given

by

$$T = \frac{b}{U^{\frac{1}{\alpha}}} \tag{2}$$

2.2 Intrusion Detection Techniques

Intrusion Detection Systems fall into two pairs of categories: host-based or network-based; and anomaly-based or signature-based.

A Host-based Intrusion Detection System (HIDS) consists of an agent on a host that identifies intrusions by analyzing system calls, application logs, file-system modifications (binaries, password files, capability databases, access control lists, etc.) and other host activities and state. In a HIDS, sensors usually consist of a software agent. Some application-based IDS are also part of this category. An example of a HIDS is OSSEC, developed by Daniel Cidd [20].

Conversely, a Network Intrusion Detection System (NIDS) is an independent platform that identifies intrusions by examining network traffic and monitors multiple hosts. Network intrusion detection systems gain access to network traffic by connecting to a network hub, network switch configured for port mirroring, or network tap. In a NIDS, sensors are located at choke points in the network to be monitored, often in at network borders. Sensors capture all network traffic and analyze the content of individual packets for malicious traffic. An example of a NIDS is Snort, developed by Martin Roesch and maintained by Sourcefire Inc. [1].

An Anomaly-Based Intrusion Detection System works by detecting computer intrusions and misuse by monitoring system activity and classifying it as either normal or anomalous. Typically, these systems begin by determining normal operating conditions for bandwidth, protocols, ports and device connections. The classification is based on heuristics or rules, rather than patterns or signatures, and will detect any

type of misuse that falls out of normal system operation. This opposes signature based systems which can only detect attacks for which a signature has previously been created.

In order to determine what is attack traffic, the system must be taught to recognize normal system activity. This is most often accomplished with artificial intelligence techniques, including neural networks and classifier systems. Another method, known as strict anomaly detection, is to first define the normal usage of the system using a strict mathematical model, and flag any deviation from this as an attack. CFEngine developed by Mark Burgess has support for this technique [12], as well as RRDTool by Tobi Oetiker [66].

Anomaly-based Intrusion Detection does have some short-comings, namely a high false positive rate and the ability to be fooled by a correctly delivered attack. Attempts have been made to address these issues through payload-based techniques used by PAYL [89] and MCPAD [69]. Signature-based systems have a very low false-positive rate, but are more limited in the types of attacks they can detect. Novel attacks which are designed to thwart signature-based systems may still be detectable by an anomaly-based system.

Some terminology and important concepts for IDSs are as follows [91]:

- *Alert/Alarm*: A signal suggesting that a system has been or is being attacked.
- *True Positive*: A legitimate attack which triggers an IDS to produce an alarm.
- *False Positive*: An event signaling an IDS to produce an alarm when no attack has taken place.
- *False Negative*: A failure of an IDS to detect an actual attack.
- *True Negative*: When no attack has taken place and no alarm is raised.

- *Noise*: Data or interference that can trigger a false positive.
- *Site policy*: Guidelines within an organization that control the rules and configurations of an IDS.
- *Site policy awareness*: An IDS's ability to dynamically change its rules and configurations in response to changing environmental activity.
- *Confidence value*: A value an organization places on an IDS based on past performance and analysis to help determine its ability to effectively identify an attack.
- *Alarm filtering*: The process of categorizing attack alerts produced from an IDS in order to distinguish false positives from actual attacks.
- *Attacker or Intruder*: An entity who tries to find a way to gain unauthorized access to information, inflict harm or engage in other malicious activities.
- *Masquerader*: A user who does not have the authority to a system, but tries to access the information as an authorized user. They are generally outside users.
- *Misfeasor*: They are commonly internals who misuse their powers
- *Clandestine user*: A user who acts as a supervisor and tries to use his privileges so as to avoid being captured.

2.3 Flow-based Intrusion Detection

The traditional idea of a network *flow*, as defined in [95], is a unidirectional data stream between two computer systems where all transmitted packets of this stream share the following characteristics: IP source and destination address, source and destination port, and IP protocol. Thus, all network packets sent from host A to

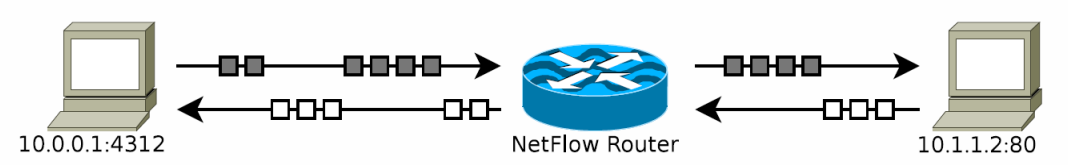


Figure 2. Network flow

host B sharing the above mentioned characteristics form a flow. Every communication attempt between two computer systems triggers the creation of a flow, even if no connection is established. In the simplest case, a complete flow is well-defined when a complete flow set-up and tear-down are observed, as is the case with most TCP communications. Complexity in any flow definition occurs when the set-up is incomplete or tear-down is abnormal. UDP is notoriously troublesome because it is connectionless protocol.

In addition to the above mentioned core characteristics, several other properties of a flow can be conveyed, for instance:

- The number of packets which have been transferred
- The number of bytes which have been transferred
- The start or end time of a flow
- The disjunction of all TCP flags occurring in the flow

Figure 2 illustrates a bidirectional communication between two computers which results in the creation of two flows. Host A is the initiator of the communication and has the IP address 10.0.0.1. Host A sent several packets to host B which is assigned the IP address 10.1.1.2. The source port of this communication is 4312 on host A whereas the destination port is 80 on host B. All the network traffic is monitored by the NetFlow router. The communication finally results in two unidirectional network flows. The first flow (illustrated as grey squares) describes the communication from A to B and the second flow (illustrated as white squares) from B to A.

Winter [95] describes a technique to collect network flows on actual hardware, with a commercial package called NetFlow. NetFlow runs on Cisco routers and collects flow statistics which it sends to a central collector. A separate device can poll this collector to run analysis on current flows in the network. MFIRE does not use live network flows – instead all traffic is simulated. However the NetFlow architecture provides a well-known framework for modeling flows, and this model is useful in our discussions.

A useful set of real-world flow data and metrics is provided by Andrew Moore [65]. Real network traffic was collected over a 24-hour period at a research facility with approximately 1000 active workstations. Individual flows are constructed from this data, and labeled as *idle*, *interactive* (two-way), or *bulk* (one-way). Only data and metrics corresponding to the TCP protocol are collected; UDP and ICMP are ignored. Flows are characterized into 249 metrics.

However, one does not need to observe a specific TCP connection or tear-down to use flows. A *microflow* abandons such concepts in favor of observing traffic in a more immediate fashion. This concept treats flows as a collection of packets to/from nodes, but does not distinguish bi-directional flows; everything is treated as one-directional. These flows are robust to incorrectly formatted TCP connections and tear-downs because they do not rely on those actions for measurement. A disadvantage is that microflows lose potentially useful information, including the cumulative time that a connection has been established, or the amount of data sent since the beginning of a connection. A good comparison between the usefulness of both approaches for the ID problem is provided in [88].

In the environment used in this research, TCP is not specifically implemented; rather everything behaves like UDP. Because of this, mircroflows are the obvious choice. However, future work should examine the use of TCP and full connection-

oriented flows.

2.4 Taxonomy of Attacks

This section introduces three common types of network attacks: distributed denial of service, vulnerability Scans and worm propagation. We focus on attacks which cause significant changes in traffic flows, since this is framework for the current research. Background on other attacks can be found in [80].

2.4.1 Denial of Service Attacks.

Mirkovic [64] presents a comprehensive taxonomy of different DDoS attack types, Figure 3. For this research, we concentrate on brute-force attacks, also called flood attacks [83], although the MFIRE environment is capable of simulating any other model. A flood attack involves malicious agents sending large volumes of traffic to a victim system, to congest the victim system's network bandwidth with IP traffic. The victim system slows down, crashes, or suffers from saturated network bandwidth, preventing access by legitimate users. Flood attacks can be executed using both User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) packets.

Formal models for DDoS and their detection are proposed in the several articles. [63] applied to DDoS detection the k-nearest neighbor (kNN) algorithm improved by feature weighting and selection based on a genetic algorithm. Overall accuracy of over 97% for known DDoS attacks is achieved, and over 78% in the case of unknown attacks.

Scepanovic [76] focuses on the scenario in which a cluster-based filter is deployed at the target network and serves for proactive or reactive defense. A game-theoretic model is created for the scenario, making it possible to model the defender and attacker strategies as mathematical optimization tasks. The model is based on the

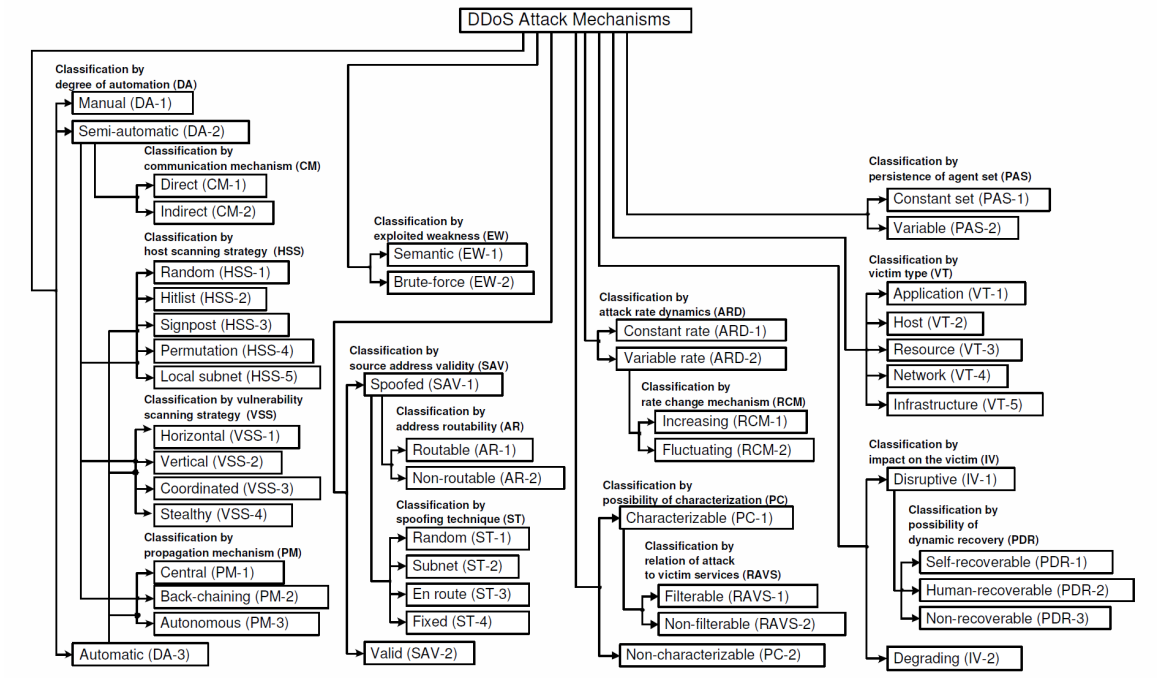


Figure 3. Taxonomy of DDoS Attack Mechanisms

continuous nonlinear knapsack problem. The experimental outcome shows the high effectiveness of cluster-based filtering in proactive and reactive DDoS defense.

If the DoS attack can be detected eventually, a common question is why do we need attack detection [68]? There are three reasons for attack detection. First, if a target can detect an attack before the actual damage occurs, the target can win more time to implement attack reaction and protect legitimate users. Second, if attacks can be detected close to attack sources, attack traffic can be filtered before it wastes any network bandwidth. However, there is generally insufficient attack traffic in the early stage of an attack and at links close to attack sources. Consequently, it is easy to mistake legitimate traffic as attack traffic. Therefore, it is challenging to accurately detect attacks quickly and close to attack sources. Finally, *flash crowds* are very similar to DoS attacks, which can cause network congestion and service degradation. However, flash crowds are caused by legitimate traffic, whereas DoS attacks caused by malicious traffic. Hence, it is important to differentiate DoS attacks from flash

crowds so that targets can react to them separately.

DoS attacks can be easily detected since the target's services will be degraded, for example, with a high packet drop rate. Second, false positives are a serious concern for DoS attack detection. Since the potency of DoS attacks does not depend on the exploitation of software bugs or protocol vulnerabilities, it only depends on the volume of attack traffic. Consequently, DoS attack packets do not need to be malformed, such as invalid fragmentation field or malicious packet payload, to be effective [68]. As a result, the DoS attack traffic can look very similar to legitimate traffic.

2.4.2 Vulnerability Scans.

A vulnerability scan can be used to conduct network reconnaissance, which is typically carried out by a remote attacker attempting to gain information or access to a network on which it is not authorized or allowed. Network reconnaissance is increasingly used to exploit network standards and automated communication methods. The aim is to determine what types of computers are present, along with additional information about those computers; such as the type and version of the operating system. This information can be analyzed for known or recently discovered vulnerabilities that can be exploited to gain access to secure networks and computers. Network reconnaissance is possibly one of the most common applications of passive data analysis. Numerous tools exist to make reconnaissance easier and more effective.

A port scan is an attack that sends client requests to a range of server port addresses on a host, with the goal of finding an active port and exploiting a known vulnerability of that service [78]. The result of a scan on a port is usually generalized into one of three categories:

- *Open*: The host sent a reply indicating that a service is listening on the port.
- *Closed*: The host sent a reply indicating that connections will be denied to the

port.

- *Filtered*: There was no reply from the host.

Potential security concerns exist for both the program responsible for delivering a service (on open ports), and with the operating system that is running on the host (on open or closed ports). Filtered ports do not tend to present vulnerabilities. There are many standard scanning formats, some of which follow standard Internet protocols, others which (purposefully) do not [80]. Some common techniques are outlined here:

TCP CONNECT scan—The simplest port scanners use the operating system's network functions. If a port is open, the operating system completes the TCP three-way handshake, and the port scanner immediately closes the connection. Otherwise an error code is returned. This scan mode has the advantage that the user does not require special privileges. However, using the OS network functions prevents low-level control, so this scan type is less common. This method is noisy, particularly if it is a complete sweep of all ports: the services can log the sender IP address and Intrusion detection systems can raise an alarm.

TCP SYN scan—SYN scan is another form of TCP scanning. Rather than use the operating system's network functions, the port scanner generates raw IP packets itself, and monitors for responses. This scan type is also known as "half-open scanning", because it never actually opens a full TCP connection. The port scanner generates a SYN packet. If the target port is open, it will respond with a SYN-ACK packet. The scanner host responds with a RST packet, closing the connection before the handshake is completed.

The use of raw networking has several advantages, giving the scanner full control of the packets sent and the timeout for responses, and allowing detailed reporting of the responses. SYN scan has the advantage that the individual services never actually receive a connection. However, the RST during the handshake can cause problems

for some network stacks, in particular simple devices like printers.

UDP scan—UDP is a connectionless protocol so there is no equivalent to a TCP SYN packet. However, if a UDP packet is sent to a port that is not open, the system will respond with an ICMP port unreachable message. Most UDP port scanners use this scanning method, and use the absence of a response to infer that a port is open. However, if a port is blocked by a firewall, this method will falsely report that the port is open. If the port unreachable message is blocked, all ports will appear open. This method is also affected by ICMP rate limiting.

An alternative approach is to send application-specific UDP packets, hoping to generate an application layer response. For example, sending a DNS query to port 53 will result in a response, if a DNS server is present. This method is much more reliable at identifying open ports. However, it is limited to scanning ports for which an application specific probe packet is available. Some tools (e.g., nmap) generally have probes for less than 20 UDP services, while some commercial tools (e.g., nessus) have as many as 70. In some cases, a service may be listening on the port, but configured not to respond to the particular probe packet.

TCP ACK scan—ACK scanning is one of the more unique scan types, as it does not exactly determine whether the port is open or closed, but whether the port is filtered or unfiltered. This is especially good when attempting to probe for the existence of a firewall and its rulesets. Simple packet filtering will allow established connections (packets with the ACK bit set), whereas a more sophisticated stateful firewall might not.

TCP FIN scan—Firewalls are, in general, scanning for and blocking covert scans in the form of SYN packets. FIN packets are able to pass by firewalls with no modification to its purpose. Closed ports reply to a FIN packet with the appropriate RST packet, whereas open ports ignore the packet on hand. This is typical behavior

due to the nature of TCP, and is in some ways an inescapable downfall.

2.4.3 Worms.

It is vital to detect active worms effectively. In the near future active worms may spread across the whole Internet in a very short period of time, making the average detection time critical. A common way to detect worms is to place sensors in a network to monitor messages sent to non-existent IP addresses. Administrators of networks are aware of exactly which IP addresses are in use in their domains, and common worm attacks do not have access to this information. If a message is sent to a non-existent IP, then this flags the sender as suspicious [17]. Attackers that wish to build stealth into the system must take preliminary steps to discover a network map prior to initiating the worm.

Many models exist for worm propagation [17, 55, 87, 99, 100, 49, 77]. The basis of many of these is the *general epidemic* model, which considers a fixed population size N where each individual can be in one of three states: susceptible to the disease (S), infected (I), or removed (R) [55]. In networking terms, removals can occur if the victim is taken offline or becomes immune (patched) to the infection. The normal state progression for an individual is $S \rightarrow I \rightarrow R$, normally termed an SIR model. But in the networking domain, victims who recover and do not obtain immunity to the infection will again become susceptible: $S \rightarrow I \rightarrow S$, an SIS model. Also known as the *Epidemiological Model*, this is formally represented as:

$$\frac{dn}{dt} = \beta(1-n) - dn \tag{3}$$

where $n(t)$ is the fraction of infected nodes, β is the infection parameter, and d is the

Table 1. Parameters for the spread of active worms

vulnerable machines	N	number of vulnerable machines
Size of hitlist	h	number of infected machines at the beginning of the spread of active worms
Scanning rate	s	average number of machines scanned by an infected machine per unit time
Death rate	d	rate at which an infection is detected on a machine and eliminated without patching
Patching rate	p	rate at which an infected or vulnerable machine becomes invulnerable

death rate. The solution to the above equation is

$$n(t) = \frac{n_0(1-\rho)}{n_0 + (1-\rho-n_0)e^{-(\beta-d)t}} \tag{4}$$

where $\rho = \frac{d}{\beta}$ and $n_0 \equiv n(t=0) = \frac{sizeofhitlist}{N} = \frac{h}{N}$

2.5 Pattern Recognition

In machine learning, pattern recognition is the assignment of a label to a given input value [75]. An example of pattern recognition is classification, which attempts to assign each input value to one of a given set of classes. However, pattern recognition is a more general problem that encompasses other types of output as well. Other examples are regression, which assigns a real-valued output to each input; sequence labeling, which assigns a class to each member of a sequence of values; and parsing, which assigns a parse tree to an input sentence, describing the syntactic structure of the sentence.

Pattern recognition algorithms generally aim to provide a reasonable answer for all possible inputs and to do "fuzzy" matching of inputs. This is opposed to pattern matching algorithms, which look for exact matches in the input with pre-existing patterns. Algorithms for pattern recognition depend on the type of label output,

on whether learning is supervised or unsupervised, and on whether the algorithm is statistical or non-statistical in nature.

2.5.1 Classification.

In pattern recognition, we want to learn $x \mapsto y$ where $x \in X$ is an object and $y \in Y$ is a class label. For a 2-class problem, we have $x \in R^n, y \in \pm 1$. Given a training set $(x_1, y_1)...(x_m, y_m)$, we want to train the classifier to generalize such that given a previously seen $x \in X$ it finds a suitable $y \in Y$. In other words, we want to find a classifier $y = f(x, \alpha)$ where α are the parameters of the function. If we are choosing our model from the hyperplanes in R^n then we have

$$f(x, w, b) = \sinh(w \cdot x + b) \tag{5}$$

We can attempt to learn $f(x, \alpha)$ by choosing a function that performs well on training data:

$$R_{emp}(\alpha) = 1/m \sum_{i=1}^{m} l(f(x_i, \alpha), y_i) \tag{6}$$

where l is the zero-one *loss function*, $l(y, \hat{y})$ if $y \neq \hat{y}$ and 0 otherwise. R_{emp} is called the *empirical risk*, and represents the *training error*.

We are trying to minimize the overall risk:

$$R(\alpha) = \int l(f(x, \alpha), y) dP(x, y) \tag{7}$$

where $P(x, y)$ is the (unknown) joint distribution function of x and y. $R(\alpha)$ represents the *test error*.

Most methods used for classification use numerical values and are unable to handle symbolic information directly. In the intrusion detection problem, packet data can be highly qualitative in nature. Many flags are present in packet headers that are

non-numerical, but which may be beneficial in detecting an attack. The challenge lies in converting this information to a form which a classifier can interpret. Experiments conducted in [39] compare the effectiveness of three different symbolic conversion methods. Results demonstrate that these three methods improve the prediction ability of a classifier, with respect to the arbitrary and commonly used assignment of numerical values. None of the indicated techniques are used in this work, but should be explored in future research.

2.5.2 Clustering.

Classification requires prototype patterns from each class. Algorithms that derive the decision or discriminant function using prototype patterns or training data are called supervised algorithms for learning. Clustering algorithms form a special class of algorithms that can identify natural groupings of data; and derive the class prototypes or "cluster centers." These algorithms do not need training samples and are referred to as unsupervised learning algorithms. Further, clustering algorithms are not based on the use of a discriminant function and no decision boundaries are generated.

Cluster-seeking algorithms partition a given set of patterns $x_1, x_2, ..., x_N = U$ into M disjoint sets. This is done on the basis similarity of patterns in the same class. Clustering procedures are based on first selecting an appropriate similarity measure.

A commonly used clustering algorithm in this class is the k-means algorithm, which is based on minimizing a performance index, F. K is the number of clusters specified by the user, and F is the sum of squared distances of all points in a cluster to the cluster center.

Assignment step: Assign each observation to the cluster with the closest mean:

$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\| \leq \|x_p - m_j^{(t)}\| \forall 1 \leq j \leq k\} \tag{8}$$

Update step: Calculate the new means to be the centroid of the observations in the cluster:

$$m_i^{(t+1)} = \sum_{x_j \in S_i^{(t)}} x_j \tag{9}$$

The algorithm is deemed to have converged when the assignments no longer change. In general, there is no guarantee that it will converge to the global optimum, and the result may depend on the initial clusters. As the k-means algorithm is usually very fast, it is common to run it multiple times with different starting conditions.

2.5.3 Support Vector Machines.

The concept of Support Vector Machines (SVM), as proposed by Vladimir Vapnik at AT&T Bell Laboratories, emerged from the field of statistical learning theory [53, 38]. SVMs were originally used to solve supervised two-class classification problems. Over the years researchers came up with numerous enhancements such as one-class SVMs and multi-class SVMs. They are a well-known and popular technique for classification and regression. SVMs feature a general combination of high accuracy, fast classification and fast training time. The task of supervised two-class classification is solved by determining an optimal separating hyperplane between the two given classes. Depending on whether linear or nonlinear SVMs are used, this determination can happen in a high-dimensional feature space.

Vapnik showed that an upper bound on the true risk can be given by the empirical risk plus an additional term:

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\frac{h(log(\frac{2m}{h}+1) - log(\frac{n}{4})}{m}} \tag{10}$$

where h is the dimensionality of the set of functions parameterized by α. This is

a measure of the functions' capacity or complexity. The more phenomena that are described, the larger the value of h. Therefore, h is *the maximum number of points that can be separated in all possible ways by that set of functions.*

As many other machine learning methods, SVMs operate in vector spaces. The dimension of the vector space is determined by the amount of features used. An SVM is characterized by its separating Hyperplane $f(x) = w \cdot \Phi(x) + b$ where w represents the normal vector perpendicular to the hyperplane, b represents the offset from the origin, and features are mapped to higher dimensional space with $x \mapsto \Phi(x)$. For example, a polynomial mapping is represented by

$$\Phi : R^2 \to R^3(x_1, x_2 \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)x_1x_2}, x_2^2) \tag{11}$$

A hypothetical hyperplane is illustrated by the dashed line in diagram (a) and (b) of Figure 4. The hyperplane of diagram (b) is said to separate both data sets in an optimal way since its margin to the two surrounding lines, representing the class borders, is maximized. Afterwards, the classification of a vector (i.e., a testing sample) is performed by determining on which "side" of the hyperplane the vector lies, i.e., to which class it belongs.

Diagram (a) of Figure 4 features a less optimal hyperplane. In this case the margin of the hyperplane is visibly smaller than in diagram (b). This affects the generalization ability since vectors lying very close to the hyperplane can be classified wrong. The difficulty of training an SVM now lies in finding the optimal separating hyperplane. The hyperplane is calculated from a training set (equation), where (equation) (the exponent g stands for the amount of features) and (equation).

The vectors lying closest to the hyperplane are referred to as support vectors. Only these vectors are used for calculating the hyperplane.

The dimensionality of $\Phi(x)$ can be very large, making w hard to represent in

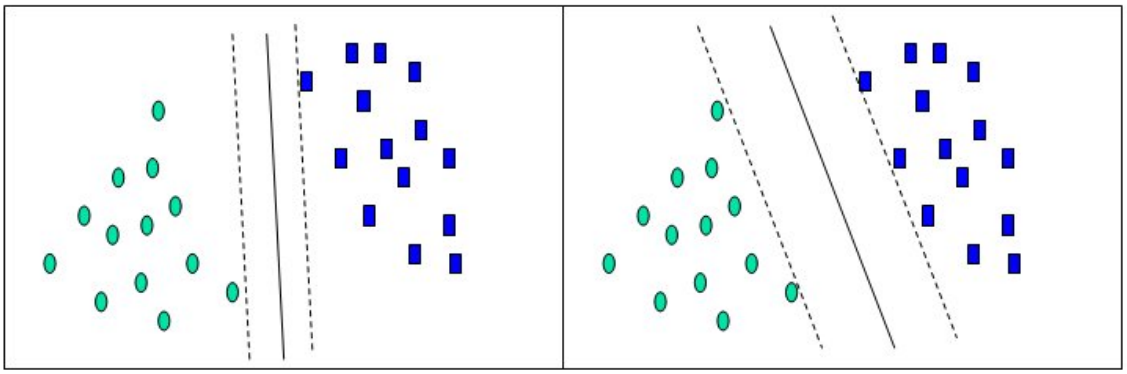


Figure 4. Poor (a) and Optimal (b) separating hyperplanes of an SVM. The poorly separating hyperplane offers bad generalization ability whereas the optimal separating hyperplane perfectly divides both data sets by maximizing the margin of the hyperplane

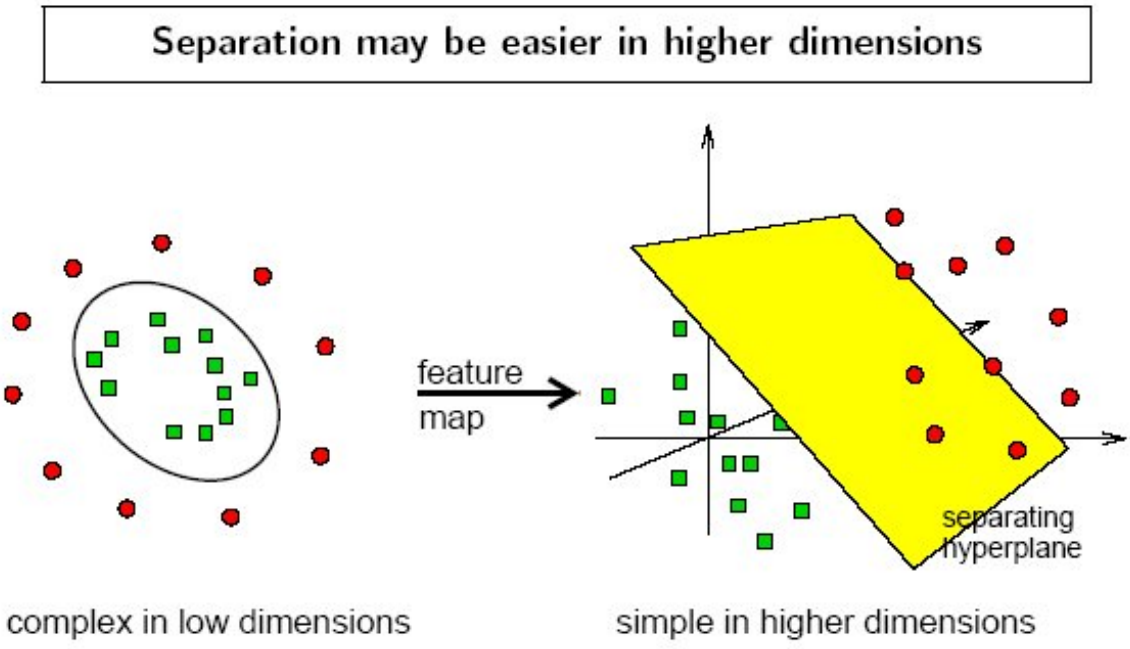


Figure 5. SVM separating features with a hyperplane in a higher dimensional space

memory, and hard to solve. Kimeldorf and Wahba (1971) presented the representer theorem, which shows that

$$w = \sum_{i=1}^{m} \alpha_i \Phi(x_i) \tag{12}$$

for some variables α. Instead of optimizing w directly we can optimize α. This gives:

$$f(x) = \sum_{i=1}^{m} \alpha_i \Phi(x_i) \cdot \Phi(x) + b \tag{13}$$

and $K(x_i, x) = \Phi(x_i) \cdot \Phi(x)$ is called the *kernel function.*

So far only the linear classification capability of SVMs is introduced. However, by means of so called kernel functions SVMs are also able to separate data which, at first glance, might not seem to be linearly separable. An example is illustrated in Figure 5. The two data sets are not linearly separable without accepting many training errors, i.e., training vectors which reside on the wrong side of the hyperplane. To solve the nonlinear classification problem, kernel functions, defined as (equation), are used the purpose of which is to transform vectors from the lower dimensional input space to the higher dimensional feature space in which the data sets become linearly separable. A kernel often recommended for first experiments is the radial basis function kernel (often also referred to as Gaussian kernel). The kernel requires one variable, namely (equation).

Here is a subset of popular SVM packages [44]:

- SVM$^{\text{light}}$ [46]: SVM$^{\text{light}}$, by Joachims, is one of the most widely used SVM classification and regression packages. It has a fast optimization algorithm, can be applied to very large datasets, and has a very efficient implementation of the leave-one-out cross-validation. Distributed as C++ source and binaries for Linux, Windows, Cygwin, and Solaris. Kernels: polynomial, radial basis function, and neural (tanh).

- LibSVM [15]: LibSVM (Library for Support Vector Machines), is developed by Chang and Lin and contains C-classification, v-classification, and ϵ-regression. Developed in C++ and Java, it supports also multi-class classification, weighted SVM for unbalanced data, cross-validation and automatic model selection. It has interfaces for Java, Python, R, Splus, MATLAB, Perl, Ruby, and LabVIEW. Kernels: linear, polynomial, radial basis function, and neural (tanh).

- SVMTorch: SVMTorch, by Collobert and Bengio, is part of the Torch machine learning library and implements SVM classification and regression. Distributed as C++ source code or binaries for Linux and Solaris.

- Weka: Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from a Java code. Contains an SVM implementation.

- SVM in R: This SVM implementation in R (http://www.r-project.org/) contains C-classification, n-classification, e-regression, and n-regression. Kernels: linear, polynomial, radial basis, neural (tanh).

- MATLAB SVM Toolbox: This SVM MATLAB toolbox, by Gunn, implements SVM classification and regression with various kernels: linear, polynomial, Gaussian radial basis function, exponential radial basis function, neural (tanh), Fourier series, spline, and B spline.

- TinySVM: TinySVM is a C++ implementation of C-classification and C-regression which uses sparse vector representation and can handle several ten-thousands of training examples, and hundred-thousands of feature dimensions. Distributed as binary/source for Linux and binary for Windows.

- Spider: Spider is an object orientated environment for machine learning in

MATLAB, for unsupervised, supervised or semi-supervised machine learning problems, and includes training, testing, model selection, cross-validation, and statistical tests. Implements SVM multi-class classification and regression.

- jlibsvm [81]: Heavily refactored Java port of LibSVM. Implements optimized kernel functions using Java class structure and APIs, and has support for multithreaded training.

SVMs have shown good results in data classification; however their training complexity is very dependent on the size of the dataset. SVMs are known to be at least quadratic with the number of training data points. One approach to reduce training data size is to use a hierarchical clustering algorithm, as described by Horng [42]. That algorithm creates a clustering feature tree, which is then used to merge disjoint clusters. Experiments using this technique on the intrusion detection problem are encouraging [42].

2.6 Multiagent Systems

A common design question for any IDS is how to maximize the benefits and minimize the penalties associated with network-based as well as host-based approaches. The Multiagent System (MAS) paradigm offers a way to accomplish this, with the added advantages of flexibility and robustness provided by this approach.

Russell and Norvig [75] define a single agent through several properties: autonomous operation, ability to perceive the environment, persistence over a long period of time, ability to adapt to change, and ability to create and pursue goals. These goals are typically in support of a broader objective. Franklin and Graesser [31] provide a survey of definitions for software agents, and an associated taxonomy.

Multiagent systems can be used to solve problems that are difficult or impossible for an individual agent or a monolithic system to solve. A multiagent system is a

collection of agents that collaborate, explicitly (e.g., via cooperation) or implicitly (e.g., via competition) to achieve a broad objective or series of objectives. The main feature which is achieved when developing multi-agent systems is flexibility, since a multi-agent system can be added to, modified and reconstructed, without the need for detailed rewriting of the application. These systems also tend to be rapidly self-recovering and failure proof, usually due to the heavy redundancy of components and the self-managed features.

In the networking domain, if agents are required to be *mobile*, then all hosts in the network must have a generic agent platform installed which provides the environment in which the agent executes. Agent migration then consists of sending agent state to a remote process responsible for reinstantiating the agent.

Jansen lists some specific advantages of a mobile, agent-based IDS [45]:

- *Overcoming network latency* - if an agent is present on a node requiring remedial action, the agent can respond more quickly than if action must be initiated by a central coordinator

- *Reducing network load* - Communication requirements are reduced by allowing agents to process sensor data locally, instead of requiring each node to send sets of sensor observations to a central processing location. Sharing the results of local processing incurs a relatively light demand on bandwidth.

- *Autonomous execution* - surviving agents continue to operate when part of the IDS fails

- *Platform independence* - agent platforms with standard interfaces may be written for multiple operating systems to allow effective MAS execution in a heterogeneous OS environment

- *Dynamic adaptation* - the system can be reconfigured during run-time in a variety of ways. The mobility of the agents allowing them to seek effective positions is a reconfiguration. Agents can clone themselves or request assistance from other agents in high demand situations. Selected agents can be replaced while non-selected agents continue to operate. One can also update repositories of behaviors and parameters which agents access periodically.

Potential disadvantages include decreased performance and/or increased resource consumption when mobility is implemented ineffectively. Also, since each agent is a member of a trusted network that, if compromised, could provide the attacker considerable leverage, digitally signed communications (including migrations) are essential.

2.7 Reputation

Trust and reputation are central to effective interactions in open multi-agent systems (MAS) in which agents, that are owned by a variety of stakeholders, continuously enter and leave the system. Such a concept of reputation focuses on the difficulty for agents to form stable trust relationships necessary for confident interactions. This implies an environment in which individual agents are greedy, able to make their own decisions, and not necessarily seeking to optimize the good of the system.

Many computational and theoretical models and approaches to reputation have been developed [25, 96, 79, 74, 43]. In all cases, electronics personas are created, which reflect the specific forum under evaluation (ecommerce, social networks, blogs, etc.). Most of these models are either too abstract or explicitly model an environment that does not apply to the one presented here. In particular, concepts such as social networks, past experiences with agents, greed, competition, sparse participation of agents, building trust over time, and coalitions all miss the mark on the approach defined in this research. Many of these concepts are good candidates for future

research, especially when paired with a fully distributed MAS.

The first objective of this research uses a different definition of reputation. In the system described, agents do not make decisions on their own. Agents simply use the available local observations in order to make a classification, which is sent to a central agent controller. The controller dictates to individual agents whether they should move to a new location. In this sense, agents are simply the eyes of a single central controller and cannot be enticed to perform any individual actions. In other words, all agents are fully trusted. Rather, the view of reputation is of a metric to judge how capable the agent is to make an informed classification given its current location.

In the general sense, reputation is *what is generally said or believed about an [agent's] character or standing* [47].

One example of a multiagent system using Reputation is SPORAS [96], in which new agents start with a minimum reputation value, and build up reputation during their time on the system. Reputation is raised or lowered based on feedback from other parties. In addition, agents' reputation is never allowed to fall below the level of a new user, thus preventing an agent from purposely exiting and re-entering the system just to improve its reputation. Our design does not have that issue, but SPORAS represents some of the concepts that we are seeking. Other research contained in [43, 74, 79] demonstrate other approaches.

We can consider the concept of Reputational Incentives defined in [13]: the trustor calculates the reputational gain (or damage) that a trustee will experience as a result of good (or bad) feedback being communicated to the society, and considers this as an additional incentive.

Pertaining to the intrusion detection problem, a trust model is defined by [10], and also makes use of the NetFlow concept of flows. The defined trust model is a

specialized knowledge structure which is used by the agents in the second stage of the processing. It aggregates the anomalies provided by all agents, and integrates the anomalies of current flows with the similar flows observed in the past. The similarity of flows is based on comparison of their features. Each network flow can be described by a set of relevant observable features (feature vector). These features define the feature space, a metric space on which the trust model of each agent operates. Trustfulness is determined for significant clusters (significant flow samples) in this space, and the anomaly of each flow is used to update the trustfulness of centroids in its vicinity. Therefore, it reflects the past typical anomaly of similar flows in a similar situation. Each agent uses a distinct set of features to describe the flow, and the added value of the mechanism (which is similar to ensemble learning/classification) is in combining the trustfulness assessments aggregated in different trust models into a single coherent value.

2.8 Evolutionary Computation

In a MAS, there are many "controls" within an agent that define its behavior, including classifiers, sensors and movement actuators. In the networking domain, it would be beneficial to find control parameters that optimize the performance of the IDS. For example, parameters of the agent's classifier can be changed; but as the system's degrees of freedom increase, finding an optimal set of parameters quickly becomes intractable.

In this and many other application domains it is easier to recognize a good solution than to find it in the first place. Stochastic search algorithms offer a method for finding near-optimal solutions to these types of problems [85]. Search can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions. Search algorithms move from solution to solution

in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.

The simplest local search algorithm is hill climbing, which is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, that incremental change is made to the new solution. This repeats until no further improvements can be found.

Simple hill climbing suffers from a tendency to get stuck at a local maximum. Some improvements to the algorithm attempt to mitigate this tendency, and include stochastic hill climbing, random-restart hill climbing, hill-climbing with backtracking and tabu search.

Another promising alternative is Evolutionary Algorithm (EA) which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. The generalized notion of an EA is applicable to several sub-domains; chiefly, Genetic Algorithms (GAs), Evolution Strategies (ESs), and Evolutionary Programming (EP) [4, 21, 85]. EAs draw inspiration from organic evolution as a means of searching for competitive solutions in situations where efficient search for the optimal solution is elusive (e.g. NP-hard problems).

In evolutionary computation, the process involves initializing a *population* of candidate solutions, where each solution is a vector of parameters proposed for the system whose performance is being optimized. Each member of the population is evaluated by supplying the parameters to the system and measuring performance to determine the member's *fitness*. The measure of performance may be a single value (which could either represent a single objective or a weighted sum of objectives) or a vector (e.g. in the case of multi-objective optimization). Fitness values are used as *selection*

criteria to determine which members of the population should survive into the next generation. Selected members may then undergo *recombination* and/or *mutation* to produce new candidate solutions. Recombination produces offspring by combining the parameter values of "parent" solutions in some way. Mutation only involves one "parent" and simply changes specific parameter values as a means of exploring the solution space. The resulting population may be a mix of old and new solutions. All are evaluated as before, after which selection happens again, and so on for potentially many generations until some criteria is satisfied (e.g. some set number of generations have completed, some performance criteria has been met, or performance ceases to improve).

Bäck presents a useful EA formalism [4]. Optimization as a minimization of a function $f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, $M \neq \emptyset$ consists of searching for $\vec{x}^* \in M$ such that $f(\vec{x}^*) > -\infty$ and

$$\forall \vec{x} \in M : f(\vec{x}^*) \leq f(\vec{x})$$

This is easily converted when optimization requires a maximization. Regardless, the goal, usually unrealizable within time and other resource constraints, is to find the global optimum $\vec{x}^*$ for the objective function f within the feasible region M.

The formal definition of a generic, single objective evolutionary algorithm is provided in Appendix 3.1.

2.9 Simulation Environment

This section focuses on the underlying simulation framework. Specifically, we look at Discrete Event Simulation. This method views the simulation as being composed of a chronological sequence of events, each of which occurs in an instant and changes the state in the system, possibly resulting in more events being scheduled. Comprehensive

treatment of Discrete Event Simulation is given in [7].

Components of DES systems include:

- Clock - The simulation keeps track of current simulation time in appropriate measurement units, but unlike in real time simulations, time in a DES jumps from one instantaneous event to the next.
- Schedule - The set of events to handle, typically implemented as a priority queue sorted by event time.
- Random-Number Generator - pseudorandom, which is desired in order to support a rerun of a simulation with exactly the same behavior

Typical usage of a DES includes the gathering of statistics, for which facilities may be provided, and the specification of a stopping condition. As may be the case with continuous- but not real-time simulation, a discrete event simulation runs at a rate that is not tied to the real-world clock. When resources permit, simulations may be run potentially much faster than real time, which is useful for collecting large amounts of statistics. In other cases, it may be desired that simulations run much slower than real time, perhaps paused for an extensive period of time via checkpointing, which is useful for direct observation and analysis of system dynamics.

Parallelization of DES is discussed extensively in [32]. More recently, Park and Fishwick present their work using graphics processing unit-based clusters in [67].

2.9.0.1 Popular DES Engines.

Some of the more well-known DES options and their areas of emphasis are:

- OMNeT++, [86]: network simulation
- MASON, [56]: agent-based systems simulation

- CNET, [60, 59]: network simulation
- GloMoSim, [97]: large-scale wireless networks
- OPNET : network simulation
- NS2, [58]: network simulation
- PARSEC, [6]: parallelization
- SystemC : electronics systems-level modeling
- Tortuga : general DES with Java/Eclipse integration
- SimPy: general DES for Python

For this research, the MASON DES continues to be selected due to its agent-based features and tight Java integration. OPNET, OMNet++ and NS2/NS3 simulate routing at a much more detailed level than is needed for our purposes, though may be explored in future research.

2.10 SOMAS

Self Organized Multi Agent Swarms (SOMAS) was created by Eric Holloway to study the effects of a dynamic, decentralized intrusion defense system [41]. The multi agent system is formally modeled as a DEC-POMDP, a I-POMDP, and a new F(*-POMDP). Agents in the network are evolved using a multi-objective genetic algorithm. These agents have the ability to change location, instantiate other agents and delete agents, as well as various methods to modify GA chromosomes and fitness values. Also, enemy agents have additional methods of stealing or corrupting data on a node, sending denial-of-service packets, compromising a node, and others. These functions are activated by actuators, which get their input from rules and

sensors. The relationship between the sensors, rules and actuators are optimized by the GA, which allows agents to defend against threats in the network. The agents learn to defend against attacks in a number of pre-defined scenarios, including: Intrusion Elimination, Enemy Avoidance, DDoS, and Information War. The primary goal of SOMAS is to evaluate the effectiveness of self-organization and "entangled hierarchies" for accomplishing scenario objectives. One of the interesting features of SOMAS is the ability for agents to take active defensive action in the network, rather than simply passively detecting an attack. For a complete description of SOMAS, see [41].

2.11 MFIRE v1.0

MFIRE 1.0 was created by Capt David Hancock as a network simulation environment to conduct flow-based intrusion detection experiments using a reputation-based multiagent system [37, 36, 35]. One critique of SOMAS was its rudimentary implementation of network topology and routing. MFIRE 1.0 was an attempt to create a more realistic simulation environment. MFIRE 1.0 is written in Java, and makes use of the MASON DES, and TopGen network topology generator. Networks and gateway routers are simulated down to the Autonomous System level, and packet routing is a faithful implementation of the Border Gateway Protocol. Delays and packet loss are handled by the system to a reasonable level of realism, while still allowing the DES to simulate a large network at a fast rate. In addition to the network components (nodes, links and packets), the prominent high-level objects are processes, observations and classifiers. A process object allows arbitrary code to run on any node. Subclasses derived from processes are made into agents, attackers, background internet traffic, etc. In addition, each node collects flow-based observations based on the current and past network traffic. Agents create features from these observations,

which are used to classify if an attack is occurring. Agents may use any user-defined classifier. Finally a Reputation system for the MAS allows a central agent controller to rate the reliability of each agent's classifications. This system prompts the agents to move to better vantage points within the network, and imparts self-organization to the MAS. Special attention is paid to the design of the MAS communication, which allows inter-agent communication in the presence of many types of faults (see Appendix A). The provided hierarchical class structure allows the framework to be extended for many different experiment types, using an object-oriented approach.

Capt Hancock [37] presents the hypothesis that a flow-based, multi-agent network attack classifier can be made more effective by:

1. employing a reputation system to govern agent mobility
2. adding a decay factor to each agent's reputation to further spur agents to find nodes providing the most "useful" information

From this hypothesis, four objectives are defined:

1. *Develop an effective network simulation environment appropriate for the problem scope.*
2. *Validate the proper functioning of simulated malicious traffic.*
3. *Validate the proper command, control, and communications in the multi agent intrusion detection system.*
4. *Study the effects of several factors on classification accuracy.*

The first three objectives are qualitatively validated with MFIRE 1.0. The fourth is not finished, and has been incorporated into this research. In particular, MFIRE 1.0 concludes with the realization of the simulation environment (MASON DES), network topology and routing, the MAS, agent intercommunication, reputation system,

background user (Pareto) processes, and simple attacker processes (DDoS, Worm, and Scan). These components are qualitatively validated as a complete, functional system. Although the classifier framework is put in place, MFIRE 1.0 stops short of implementing an actual classifier to allow agents to identify an attack; so no final testing of the reputation system or overall performance experiment is conducted. For a complete description of MFIRE 1.0 see [37].

This chapter provides background and theory on several topics regarding multi-agent intrusion detection. Flow-based metrics, discrete event simulation, support vector classification and stochastic search are key components in our proposed model. In the next chapter we integrate these concepts into a working framework, MFIRE-2.

III. MFIRE-2 Design

An updated design is required for autonomous classification of network attacks in a live, albeit simulated, network. In this research, we build upon the MFIRE framework of Hancock [37] by adding additional attack features, implementing a new attack classification system, and adding facilities to generate training data and test the system's classification performance. In addition, we integrate features proposed by Holloway [41] to allow agents to behave in more elaborate ways. This chapter introduces these concepts in detail.

Intrusion Detection (ID) system design is an ongoing process, due to the approximation of the network environment and the reaction to it. Thus, the multi agent system paradigm, with several performance-enhancing details, is leveraged in this design in order to maximize the performance. The agents are designed to be mobile and cooperative in terms of sharing feature observations. Over a series of simulated attacks, the integrated system searches for a 'good' distribution of agents.

A recent, innovative network-based anomaly detection system is presented in [48]. The authors use a two-stage classification approach to detect novel intrusions of various types, and is shown to have good empirical performance. Many IDS systems have shown in recent years to achieve good performance with real-world traffic [10, 39, 40, 42, 45, 63, 95]. Our approach is substantially different, in that we seek a robust environment to generate *simulated* network traffic; and the goal of our research focuses on *improvement* in performance given the movement of agents, not on achieving absolute performance outright. Thus it is difficult to find existing systems to compare our approach, however many of systems previously mentioned provide a basis for our key concepts of multi-agent systems, network-based detection, anomaly-based classification, and flow-based statistics.

The design of a suitable network simulation environment involves the representa-

tion of essential network components and operations. Specifically, nodes must route traffic, generated by processes, over links with limited capacity, in a topology reflective of what is seen in the real Internet (see Section 2.1). Some of the processes represented are 'normal,' generating traffic according to distributions seen on the real Internet, while other processes represented are 'malicious,' causing congestion on network links, systematically extracting information regarding potential vulnerabilities of network nodes, and/or spreading copies of themselves to other nodes without authorization.

To enable the properties described in such a simulated network environment requires a representation of traffic as content-bearing packets, facilities for delivering these packets to specific destination processes, and facilities for instantiating a network complete with its nodes, links, processes, and properties of each (e.g., respectively routing tables, link capacities, and traffic-generation and response behaviors).

Some of the implementation described in this chapter comes directly from [37], and is repeated here for completeness. Only information which is relevant to the updated features of MFIRE-2 is provided. For additional implementation details, the reader is encouraged to read [37].

3.1 Simluation Environment

This section presents the package hierarchy providing a framework in which to place the required representations of these concepts. In addition to the network simulation environment, a multi agent classification system is designed as a set of processes, with components including agents and an agent controller. To support the agents' classification responsibilities, interfaces are designed for classification techniques and feature definitions, enabling changes in detailed implementations without requiring changes to the system architecture.

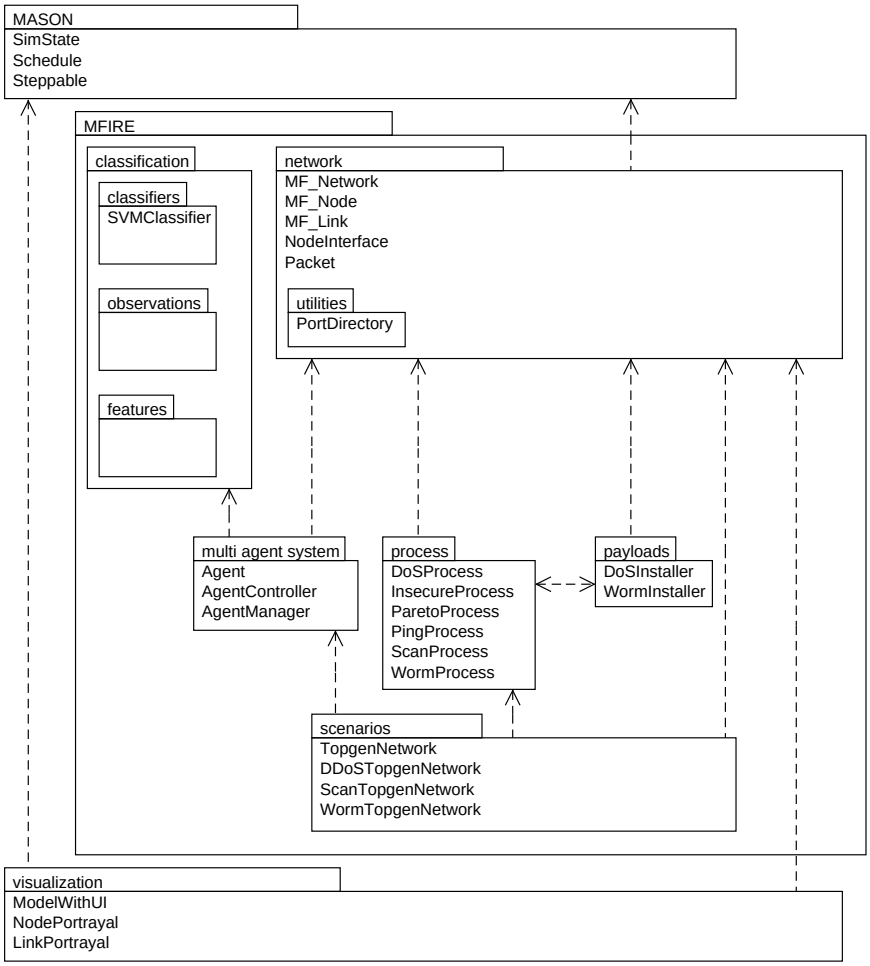


Figure 6. MFIRE package diagram.

Figure 6 presents a general view of the package hierarchy involved in the simulation. The domain layer consists of the following groups of classes:

- Network - includes representations of physical domain entities of interest. This is the 'core' of the simulation.

- Scenarios - concrete realizations of the abstract MFNetwork. The prominent class is the TopgenNetwork, which includes facilities for loading a network produced by the Topgen AS-level Internet topology generator. Each class in this package is characterized by a a unique set of Processes initially running on a subset of the nodes.

- Processes - These are analogous to the networked applications on the real Internet. Each Process runs on a host node and may receive and/or generate traffic.

- Payloads - Specially crafted payloads execute code when opened by a certain receiving processes. These payloads can be written for legitimate purposes, such as Remote Procedure Calls (RPC), but our focus is on payloads that install malicious processes on the receiving node.

- Multi agent system - This package includes the "worker bees" - the Agents, the "queen bee" - the AgentController, as well as AgentManagers with special local oversight of any Agents on the same host node.

- Classification - Agents make use of entities in this package to make local classification decisions. Included are the classification algorithms, enclosed in the 'classifiers' package, and the observations and features used. Strictly speaking, both observations and features are statistics-based calculations, but we distinguish the observations as being more "raw" than the features. By 'feature' we

imply there is something composite in its nature - it may be an average of observation values or the result of some other series of mathematical operations on the observations and/or other features.

At the top of Figure 6 is the MASON discrete event simulation engine package, which provides many vital facilities for the execution of the simulation as well as the visualization of the same. The details of the visualization are specified via entities in the visualization package at the bottom of the diagram.

Figure 7 provides a class diagram for some architectural detail of the more prominent aspects of the domain representation. This is not intended to provide a comprehensive listing of the classes nor the attributes and methods of each class. Rather, expressed are some of the essential class associations and hierarchies that drive the network simulation.

3.1.1 Network design.

The physical network components simulated in this research investigation include [37]:

- Nodes - each node represents an Autonomous System (AS). Internal to an AS is a collection of routers, switches, firewalls, and edge devices, including servers and clients. These devices are all abstracted into one node in our simulation, represented by the MF_Node class in Figure 7. Nodes route traffic via routing tables, initialized via the Floyd-Warshall shortest path algorithm [29]. This is analogous to gateway routers employing BGP on the real Internet, though with BGP, policy decisions often trump routing efficiency (competing Internet service providers, for example, may refuse to allow 'through' traffic without compensation). Each node is addressable by a unique identification number. Nodes provide resident processes with basic communications facilities, such as

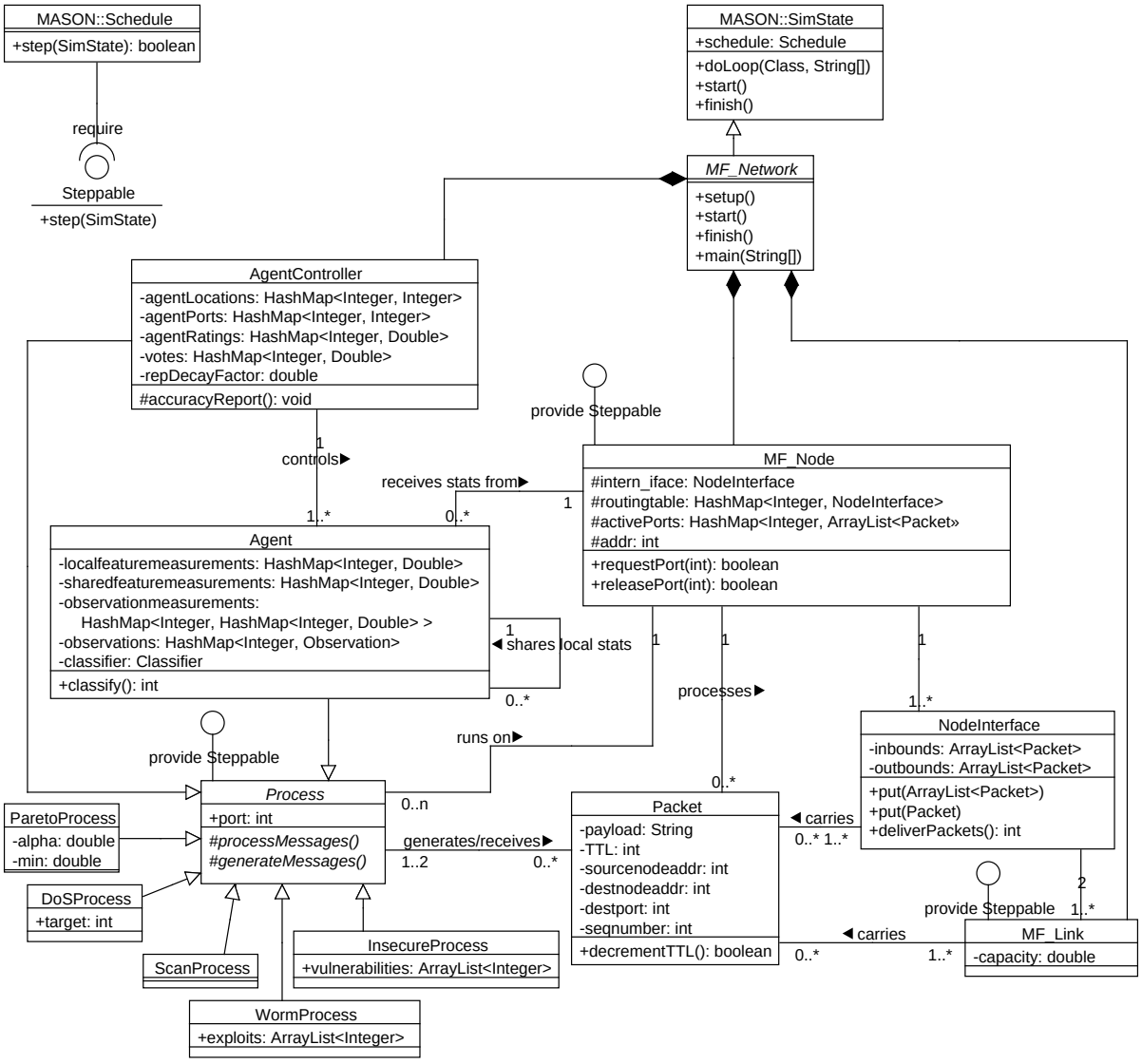


Figure 7. MFIRE class diagram.

the `send()` method, which creates and sends packets. Nodes implement the `Steppable` interface and therefore supply a `step()` method invoked on each timestep of the simulation. This method primarily switches packets from the inbound queues of all NodeInterfaces to the outbound queues of NodeInterfaces identified in the routing table, via lookup on the packet's destination address.

- NodeInterfaces - These are intermediaries between Nodes and 1) Links; or 2) Node's resident Processes. The first case includes all external-facing interfaces, while the second describes the Node's internal interface. Each is an entry/exit point. All NodeInterfaces have an inbound queue and an outbound queue. The inbound queue is read by the attached Node and written to by the attached link. The outbound queue is read by the attached link and written to by the attached Node.

- Ports - associated with nodes, ports are the communication end points for processes running on servers and clients. In the real world, each computer typically has many thousands of ports associated with each transport-layer protocol. For example, there are 2^{16} ports available for Transmission Control Protocol (TCP) and another 2^{16} for User Datagram Protocol (UDP), the number being fixed by the width of the port field in the segment, respectively datagram header [70, 71]. In our simulation, each port on an AS node corresponds with a port on an arbitrary host internal to the AS.

- Port Directory - Certain "well-known" ports are reserved for special purposes. This is the case with the real Internet, for which a list is maintained by the Internet Assigned Numbers Authority (IANA) [3] specifies how certain ports are to be used, such as port 80 for Hyper Text Transfer Protocol (HTTP) traffic. When these standards are adhered to, finding public services is greatly

simplified. Also, filtering of certain expected types of traffic becomes simple. Observe that, in our simulation, some ports are reserved for components of the multi agent system.

- Links - links in our network simulation are strictly point-to-point and connect autonomous systems together. Links are full duplex but have finite bandwidth. Depending on the scale of the simulation, links may vary in length, affecting propagation delay. One of three scales is specified at the start of each simulation:
  - LOCAL - All links have the same unit length. Packets traverse these links in one step of simulation time.
  - REGIONAL - Link lengths vary from one to ten units. This is useful when the simulated AS topology spans a continent.
  - GLOBAL - Link lengths vary from one to 100 units. This is appropriate for simulation of an AS topology in which some of the nodes are satellites in geostationary orbits, for which propogation delays can indeed be on the order of 100 times those of terrestrial links.

  Scale is realized with each link being composed of sublinks. Links implement the `Steppable` interface. Each timestep, when the Link's `step()` method is called by the `Schedule`, the Link causes each Sublink to pass its traffic to its adjacent Sublink (or, ultimately, NodeInterface).

- Processes - these include processes that strictly generate traffic for the benefit of the simulation as well as classifying agents that generate actual communication traffic (primarily to share observations). All processes run on nodes and must be assigned a port before they can send and receive packets. Processes implement the `Steppable` interface. When `step()` is called, the Process first receives and processes traffic, and then generates outbound traffic.

- Packets - Each packet consists of the following:
  - Source node address - identifies the Node of origin
  - Source port - the port used by the sending Process
  - Destination node address - identifies the Node hosting the intended recipient Process
  - Destination port - communication endpoint for the intended recipient `Process`
  - Sequence number - Facilitates sending messages spanning multiple packets
  - TTL - Time To Live - the number of hops allowed before some intermediate Node discards the packet. This mitigates problems arising from routing loops induced by congestion or misconfiguration of the routing tables.
  - Payload - a string containing the message the sending Process wishes to pass to the intended recipient. The format of this message is entirely up to the communicating processes.
  - size - Indicates the size of the payload, in numbers of characters, if a real payload is used. If a real payload is not required (e.g. to simulate background traffic or junk traffic sent by denial-of-service processes), the sending Process can simply specify the desired size of the packet to be sent, leaving the payload string null and preserving memory.

With the previous component discussion completed, the flow of our simulation can be explained. During initialization, after all network components have been instantiated, all Processes, Nodes, and Links are scheduled to execute associated tasks on every timestep (e.g. generate traffic, process traffic, move traffic). They are prioritized as follows:

- First, Processes handle received traffic and generate new traffic

- Second, Nodes handle traffic by switching packets from inbound queues to appropriate outbound queues or ports

- Third, Links move traffic along component Sublinks toward the NodeInterfaces on either end

3.1.2 MAS design.

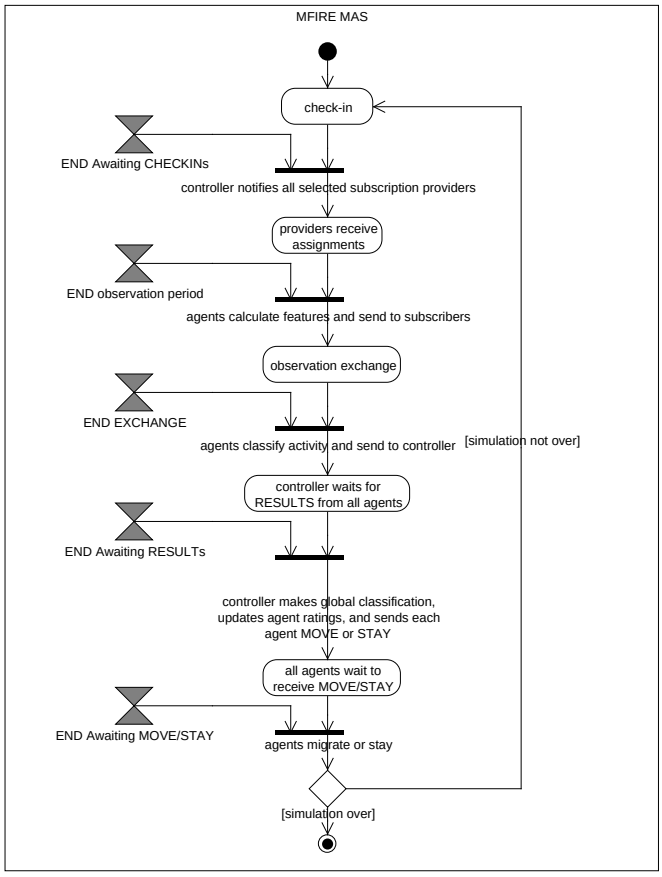
Figure 8 presents a high-level view of the nominal flow of execution from the perspective of the MAS. Five states are shown. Figure 8a indicates that the transition from each state is governed by the clock. This implies synchronization among participating elements. Typical message exchange for each state is shown in Figure 8b.

The explanation of MFIRE's high-level states is made simpler by assuming agents have been collecting observations from their respective host nodes for nearly a full cycle when it comes time to check in with the AgentController. Furthermore, each agent is assumed to have a reputation stored with the AgentController.

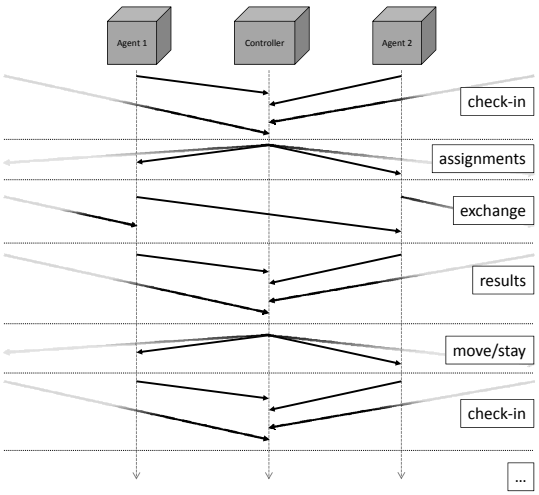
- Check-in: Agents notify the AgentController of their intention to participate in the next round of observation exchange and classification. The AgentController notes the source address and port of each CHECKIN message.

- Transition: The AgentController makes an observation sharing assignment for each Agent that checked in. It does this by constructing a roulette wheel from the reputations of other checked-in Agents. This roulette wheel is used to make a sharing assignment stochastically with preference given to Agents with higher reputations. The AgentController notifies the selected Agent with an ASSIGN message.

- Assignments: Selected Agents receive assignments. Some Agents may receive



(a) Activity Diagram



(b) Communications Example

Figure 8. MFIRE activity and client-server diagrams showing the system's normal flow of execution

multiple sharing assignments, while others receive none. For each assignment received, the Agent stores the address and port for the designated recipient as contained in the ASSIGN message.

- Transition: End the current observation cycle, calculate features, and start a new observation cycle. Observations are traffic statistics collected on each timestep. At the end of each observation cycle, there exists an Observation set for each traffic statistic measured. Features typically summarize one or more of these Observation sets. Agents calculate Feature values and store them for later use. Any Agents with sharing assignments also send their set of Feature values to all assigned recipients using SHARE messages.

- Observation Exchange: Agents wait to receive SHARE messages. Each Agent expects to receive one.

- Transition: Agents use two classifiers to make two classifications for the network activity observed over the previous cycle. One of these uses only locally calculated feature values, while the other uses the combined set of local and received feature values. Agents send the results to the AgentController in a RESULTS message.

- Results: The AgentController receives RESULTS messages from all checked-in Agents.

- Transition:

  - The AgentController tallies the votes. In each RESULTS message, the vote is the classification made using the combined local and shared feature value sets. When this is not available because the Agent never received a SHARE message, the AgentController uses the classification made using

Algorithm 1 MAS Classification

denote classification by agent a_i at time t using only local feature values as l_{it}
denote classification by agent a_i at time t using combined local and shared feature values (e.g. from peer agent a_j) as c_{it}
denote the majority classification at time t as m_t
denote network activity classes as $\mathcal{A}_k \in \mathcal{A}$ for $1 \leq k \leq K$
denote the vote tally for network activity class $\mathcal{A}_k$ at time t as v_{kt}
Require: $0 \leq \theta_l \leq 1$
procedure MASClassification(θ_l)
for all received RESULTS messages $results_{it}$ **do**
if $results_{it}$ contains a combined classification c_{it} **then**
add 1 to the vote tally v_{kt} for $\mathcal{A}_k$ for $k = c_{it}$
else
add θ_l to the vote tally v_{kt} for $\mathcal{A}_k$ for $k = l_{it}$
end if
end for
$m_t = k : v_{kt} = \max_h v_{ht}$ where $1 \leq h \leq K$
return m_t
end procedure

only the local feature value set, weighted for less influence. The system's classification is the majority vote. See Algorithm 1, in which θ_l represents the weight of a classification derived from local feature values only.

– The AgentController updates each Agent's reputation. For each Agent, each sharing assignment it had garners a rating which can positively or negatively affect the reputation. Every Agent furthermore has its reputation decayed regardless of whether it had a sharing assignment, and regardless of whether it checked in. See Algorithm 2.

– The AgentController sends each Agent a STAY or a MOVE instruction based on whether the Agent's reputation is above or below a threshold.

- Wrap-up: Agents wait to receive MOVE or STAY. Upon receiving MOVE, an Agent selects a neighboring node at random and sends a MIGRATE message to the node's AgentManager.

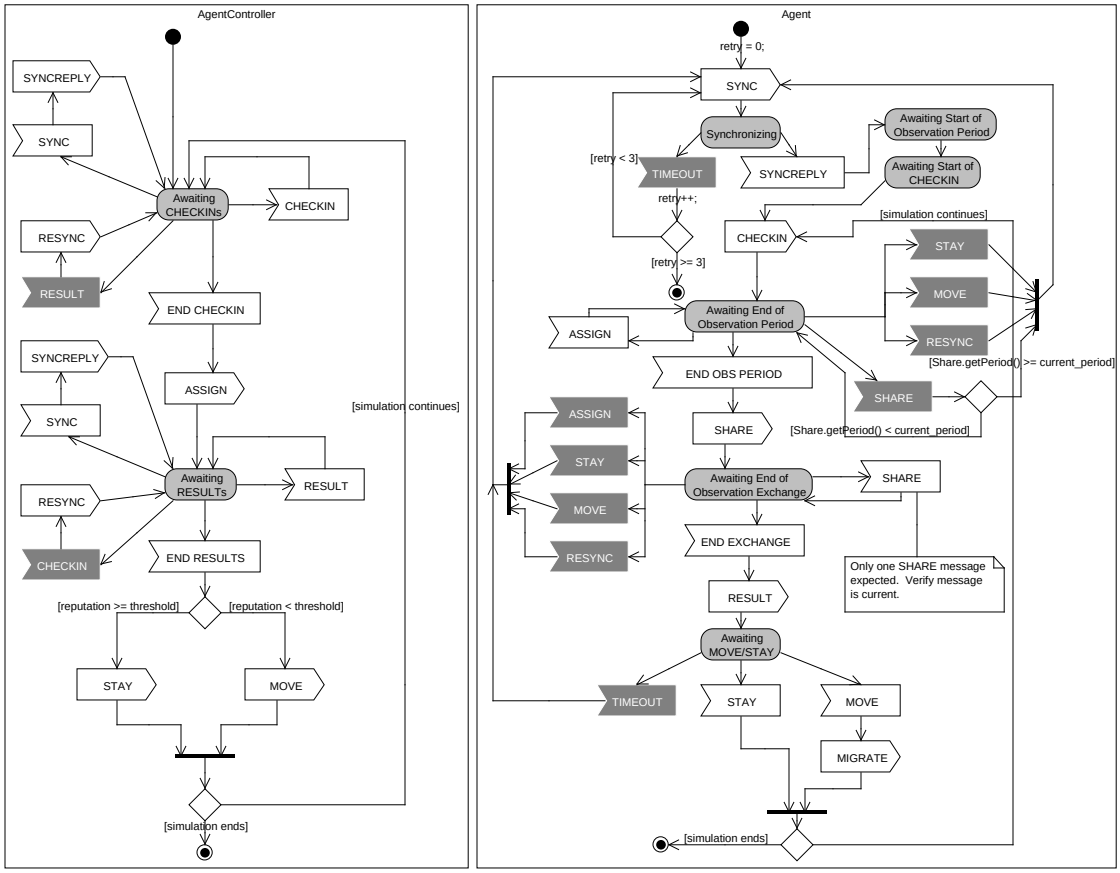


Figure 9. MFIRE detailed activity diagrams for the controller and the agent

Table 2. Comparison of Iterations 1 and 2

	Iteration 1	**Iteration 2**
AS Network Scale	'local' only	'local', 'regional', 'global'
AS Network Size	10 nodes	100 nodes
AS Network Topology	manually designed	produced by Internet topology modeler
Node Behavior	restricted processing capacity / shut down under heavy load	unrestricted processing capacity
Packet Payloads	simulated quantity only	payloads implemented and used for interprocess communication
Attacks	DoS	DDoS, Worm, Scan
MAS classifier	minimum euclidean distance	support vector machine
MAS communications	out-of-band, instantaneous	in-band with network-based delays
Feature Selection	wrapper method in a MOEA	none
MAS Objective	identify source and target of DoS attack	identify type of attack

Figure 9 shows the flow of execution of the Agent and the AgentController independently. From this figure it can be deduced that the AgentController has merely two states: it is either waiting for Agents to check in, or it is waiting for the Agents to send their results, with significant actions taking place on the transitions between states as described above. Meanwhile, the Agent has a collection of synchronization-related states, and three of the nominal states described above. It is either waiting for an ASSIGN message from the AgentController, or it is waiting for a peer to send a SHARE message, or it is waiting for a MOVE or STAY message from the AgentController.

Table 2 summarizes the key differences between this research (Iteration 2) and the

system that is quantitatively tested by David Hancock [37] (Iteration 1). In general, MFIRE features more realistic networking than in previous experimentation.

3.1.3 Observations and Features.

Each observation in MFIRE represents a traffic statistic collected over the duration of a single time period. These are used to derive feature values, which are the average and standard deviation of the observations within one observation period. We take inspriation for flow metrics from both Cisco NetFlow [95] and Moore [65], with emphasis on implementing metrics applicable to *microflows* (see Section 2.3). The fourteen metrics defined here represent a good cross-section of possible flow-based statistics, but future work should examine additional metrics, including implementing a macroflow approach (see Section 2.3 and [65]).

The fourteen observations collected by agents in MFIRE:

1. Average number of bytes per $< destaddr, destport >$-tuple
2. Average number of bytes per $< sourceaddr, sourceport >$-tuple
3. Number of distinct destination addresses
4. Number of distinct $< destaddr, destport >$-tuples
5. Number of distinct destination ports
6. Ratio of destination ports to destination addresses
7. Total number of inbound bytes
8. Total number of inbound packets
9. Ratio of packets to $< destaddr, destport >$-tuples
10. Ratio of packets to $< sourceaddr, sourceport >$-tuples

11. Number of distinct source addresses

12. Number of distinct $< sourceaddr, sourceport >$-tuples

13. Number of distinct source ports

14. Ratio of source ports to source addresses

Clearly there are many linear dependencies in this set of observations. Care must be exercised when performing feature selection from this set.

3.1.4 Attack Models.

This research consists of modeling three attacks: DDoS, worm propagation and vulnerability scan, and one normal (non-attack) mode. In all cases, background traffic is flowing on the network, and is the predominant source of packets. The attacks implemented in the current research are designed primarily to test to the effectiveness of the MAS reputation system. They are in no way a comprehensive suite of possible malware. Additional attack models should be explored in future research.

The normal (non-attack) mode consists of only background traffic. For this the Pareto model described in Section 2.1 is used with parameters $\alpha = 2.0$, and C ranges from 0.01 to 0.1, randomly selected prior to each simulation. All other attacks models also use this background traffic.

The DDoS attack consists of N processes which flood a single target T with packets to port p at rate r packets per timestep. N, T, p and r are selected randomly prior to each simulation. The node locations of the DDoS processes are random, selected from any nodes in the network.

Worm attacks are implemented by a set of vulnerable processes running on a subset of nodes in the network. A worm process is equipped with a single exploit

that targets a single vulnerability. If the exploit matches the vulnerability on the target node, the worm is able to instantiate a copy of itself on the target. Worms do not scan for vulnerabilities before attempting the attacks; they simply make an attempt. However, the worm process never sends an attack to a non-existent node. This is only possible if the worm has previously performed a scan, or otherwise been given knowledge of the current network. The current implementation assumes this knowledge is available to the worm a priori.

A vulnerability scan is modeled after simple TCP-connect port sweep. Note that the current MFIRE environment does not implement the TCP protocol explicitly, however, all processes within the environment are configured to mimic the effects of TCP and provide replies to incoming packets as needed. In particular the *connect* message may be replied with a response equivalent to an *ACK*, *ICMP port unreachable*, or ignored. The scan process runs on a single random node, which sends connection requests to a random subset of N target nodes on the network at rate r packets per timestep. The scan sends a packet to all ports in the complete range of common port numbers. N and r are randomly selected prior to each simulation.

3.2 Training the Agents

With the simulation environment set up, we can now turn our attention to agent training, in preparation for executing actual experiments. Agent training consists of generating the training data, followed by training the classifier on that data. In general, we refer to generating training data as running in *offline* mode, and testing the MAS as running in *online* mode. Many of MFIRE's functionality performs the same in both modes, and Figure 10 shows the relationship between the two. A more detailed view is provided in Figure 23 in Appendix A. In offline mode, agents do not make classifications or move; they are merely located in the network to observe and

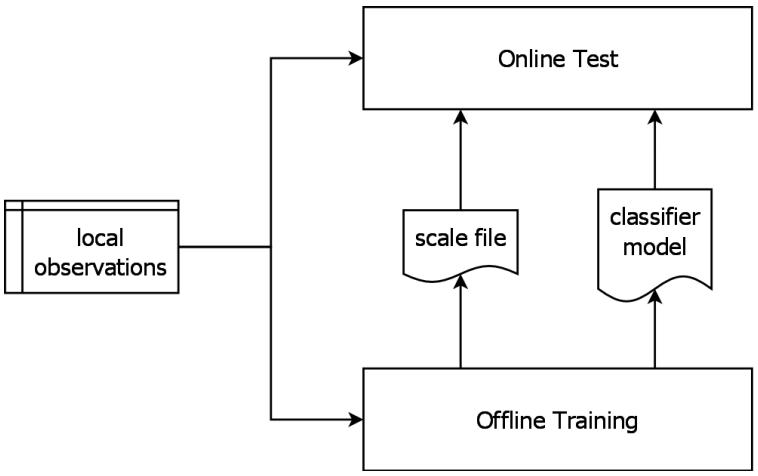


Figure 10. MFIRE offline training and online testing execution paths

log flow-based traffic statistics. A classification model and scale file are the outputs generated from this data. In online mode, agents are making active classifications and moving in the network. The agents' classifiers use the classification model and scale file as inputs.

The entire training and testing process is described in three high-level steps:

- First, generate training data (MFIRE offline mode)
- Second, train the classifier (external process)
- Third, test the MAS (MFIRE online mode)

The first two steps are conducted once. After the classifier is trained, the same one is used in all agents, for both experimental models: reputation, and free-movement. The final step is performed multiple times, as needed for the final experiments. Additionally, note that steps one and three take place within the MFIRE framework directly, and once set up do not require much user interaction. Step two requires additional user interaction, and is conducted with external software packages.

3.2.1 Generating training data.

The primary class for creating training data is contained is *DataGenerator*. This executes simulations in offline mode—that is, agents are located in fixed, random positions and do not move or generate attack classifications. In all other respects, the simulation environment behaves exactly the same way as online mode. To create training files, two agents are located in the network, and record all local flow-based statistics observed at their node. These files represent raw local data only.

After all of the required simulations are performed, several functions are applied to the local feature data. First, the two local feature files are combined to create a single combined feature file. Recall that local features and combined features are used separately, and an individual classifier is created for both. Local features from two different agents are converted into three combined features: average, multiple, and difference of the two. From this point on in the process, the local data and combined data remain distinct entities and are treated in parallel, although they are handled in the same way. Second, the feature elements are scaled to between 0 and 1, and a scale file is created, to be used later in testing. Third, the data is split into separate training and validation sets.

An additional operation to scrub the data could be added to this process as well. Scrubbing outliers has been shown to reduce training time and possibly improve classification generalization by reducing overfitting. This is not part of the current research, but should be explored in future work.

3.2.2 Training the Classifier.

The chosen classifier for this research is a Support Vector Machine (SVM) (see [53, 38]). SVM is selected due to its "high generalization performance without the need to add *a priori* knowledge," even in the face of many features [16]. Other

classifiers present many potential alternatives, and should be examined further in future research. The MFIRE environment is written to work with any classifier.

To realize an SVM implementation, the LibSVM package is selected due to its Java integration and its useful grid search method for finding optimal training parameters. LibSVM provides the needed multi-class classification technique, implemented interally as the standard one-vs-one model. We make use of the LibSVM library function *svm_predict*, standalone executables *svmtrain* and *svmscale*, and python script *grid.py*.

Feature selection is another important aspect to training a classifier. If a smaller subset of quality features can be provided to the classifier, it will be faster to train, and may improve classification generalization by reducing overfitting. One possible approach is to use Bhattacharya coefficient analysis [37]. Another simpler, albeit more computationally intensive, approach is the "leave-one-out" method. In this, the classifier is tested multiple times, each with leaving one feature out. In this way, the experimenter can see which features are useful are which are not. This method is crude in that it treats features as singular entities and does not consider the combinatorial effects they may have. A third method is to do a search (see Section 2.8) for useful features. None of these methods are part of the current research, but should be explored in future work.

3.3 Movement models

The final primary element of MFIRE is the functionality which controls agent movement. Recall that the *goal* of this research is to continue the development of a scalable software architecture for a multi-agent, flow-based intrusion detection system. The following high-level *objectives* support this goal:

- Design and evaluate a multi-agent classifier using a Reputation system
- Design and evaluate a multi-agent defense system using an Evolutionary Algo-

rithm

In this iteration of the research, the agents use the same classifier for both of the stated objectives. We are interested in comparing the performance of the MAS using two different models for the way agents move in the network. The following sections describe the reputation model and the free-movement model. These two models are compared to a baseline model of non-moving agents (fixed model). The fixed model is trivial, and is not described here in detail.

3.3.1 Agents using a reputation model.

The collective activity of the population of agents is tied together at the multi agent system (MAS) level through a controller, which processes the classification decisions ('votes') of individual agents and reports the majority result. Prior to classification, agents may receive sharing assignments from the controller, and share feature values accordingly. Each agent is then able to make a classification based on local as well as shared feature values.

The controller stores agent reputations. Each round, it calculates a rating for each sharing assignment an agent was given. The rating depends on a heuristic measure of how much the shared feature values helped or hurt the recipient's ability to classify in step with the majority. After all ratings are processed, the controller may then decay each agent's reputation by 10%. The idea is to motivate agents to explore other nodes when they are not perceived as making any positive contributions to the community.

Algorithm 2 details the idea. The values for variables *neutral*, *positive*, *bignegative*, and *smallnegative* are reflected in Table 3.

MFIRE employs a centralized reputation system per the broad categorization of [43]. This approach puts the reputation of each agent under the control of a central reputation manager. It allows simplified management of agent interactions, but is

Algorithm 2 Reputation Calculation

denote classification by agent a_j at time t using only local feature values as l_{jt}
denote classification by agent a_j at time t using combined local and shared feature values (e.g. from peer agent a_i) as c_{jt}
denote the majority classification at time t as m_t

Require: $0 < decay \leq 1$

procedure CALCULATEREPUTATIONS($decay$)
  for all agents a_i **do**
    for all recipients a_j of information provided by a_i **do**
      if $c_{jt} = m_t$ **then**
        if $l_{jt} = m_t$ **then**
          $rating_{ij} \leftarrow neutral$
        else
          $rating_{ij} \leftarrow positive$
        end if
      else
        if $l_{jt} = m_t$ **then**
          $rating_{ij} \leftarrow bignegative$
        else
          $rating_{ij} \leftarrow smallnegative$
        end if
      end if
      $reputation_i \leftarrow reputation_i + rating_{ij}$
    end for
    $reputation_i \leftarrow reputation_i \times decay$
  end for
end procedure

Table 3. How the AgentController Rates Providers of Shared Feature Values

Receiver's classification result, based on feature sets used		
Local Only	**Local + Shared**	**Rating**
Same as majority	Same as majority	+0
Same as majority	Differed from majority	-0.1
Differed from majority	Same as majority	+0.1
Differed from majority	Differed from majority	-0.05

prone to single point-of-failure issues. Future research will examine the value of a distributed approach.

Agents start with a base reputation value of 0.5 ± 0.05, which is approximately twice the migration threshold value of 0.25 used in experimentation. The *AgentController* uses Table 3 to modify reputations according to how well providers' observations helped receivers vote in step with the majority. When an agent moves to a new node, its reputation is reset to the base value. The small ± 0.05 random offset imparts some non-deterministic movement into the agents, and is needed to combat an observed behavior which causes agents to move in lockstep with each other. This occurs if agents all make the same classifications period after period, which happens when they are using a very accurate classifier that is not very sensitive to location in the network. Moving all at once is undesirable because no attack classification is given if all agents are in motion. Recall that agents can either be classifying an attack or moving, not both. In addition to this random offset, we also only allow at most 50% of the agents to move at one time. This ensures that in every period there are enough stationary agents to make a classification.

When agents vote in step with the majority, and would have done so even without the use of the shared observations, there is no reason to rate their providers positively or negatively. On the other hand, if the agent is prepared to vote in step with the majority, but ends up not doing so due to the influence of the shared observations, the

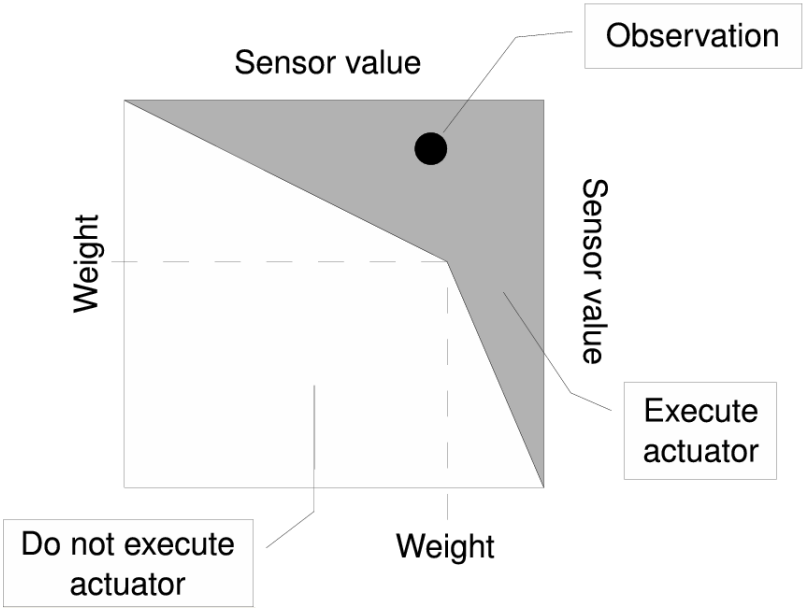


Figure 11. Classification Rule

judgment of the crowd is viewed as superior to the opinion of a single peer and thus the provider is rated negatively. Real benefit is perceived when they would have voted out of step with the majority but for the "corrective help" of the shared observations, and in such cases providers are rated positively. If the agent votes out of step with the majority and would have done so even without the shared observations, the provider is rated negatively. But, not so much as if the shared observations had dissuaded the agent from otherwise voting in step with the majority.

3.3.2 Agents using a free-movement model.

In the reputation model, agents rely on a central controller to provide move or stay commands. The controller keeps track of reputation of each agent, which is used for both sharing assignments and movement. The agents themselves do not make any local decisions. In contrast, the model introduced here decouples agent movement from the reputation system, and allows agents to move freely on their

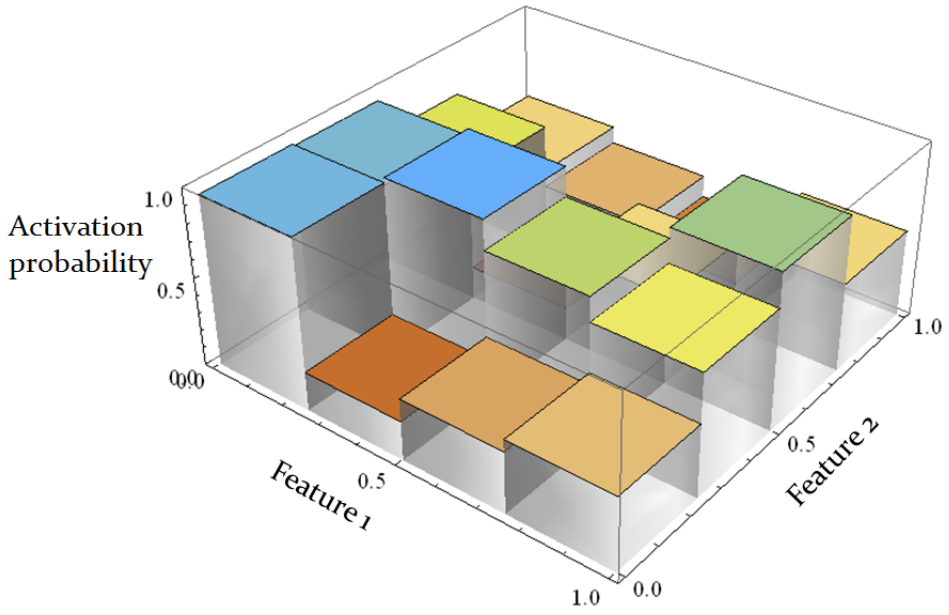


Figure 12. Movement Actuator

own. The central controller still keeps track of the reputation of each agent for feature sharing. Each agent controls its own movement locally, via an actuator which provides a binary activate or non-activate (boolean) decision. SOMAS uses this approach, and implements the actuator shown in Figure 11, which takes weights from a center point to determine a binary activation area. These weights are found using a genetic algorithm.

This research implements a similar model, but incorporates a stochastic element to the activation decision. It is important to keep agent movement in the network somewhat randomized, so that they do not cluster at the same node and never explore other areas. In addition, we like for a potential actuator to be easily manipulated with a stochastic search routine, such as genetic algorithms. The solution we propose is a probabilistic segmented actuator, shown in Figure 12. This actuator takes inputs from N features, and outputs a activation probability. Probability maps are stored directly in the actuator, and it functions as a quick lookup table, based on the location

in feature space. To develop and train such an actuator, one must define three things: dimensionality (number of features), number of segments, and choose which specific features are selected. The movement actuator may use the same local and combined feature sets available to the agents' attack classifier. For the research, we examine a 2-feature classifier, with 16 total segments. Features are selected randomly at the start of the training session. This choice makes a good reference for qualitatively testing agent behavior when using this actuator. The next chapter provides details on this as well as a full experiment on the classification capabilities of the centralized Reputation system.

IV. MFIRE-2 Experimentation and Analysis

The previous chapter describes the implementation of MFIRE-2, including the MASON discrete event simulator, networking features, attack models and SVM classification. We now turn our attention to testing that the framework operates as intended, and we perform experiments to validate the classification performance of the MAS. The next section introduces high-level concepts on experimental design. In Sections 4.2 and 4.3 we describe the test plans for meeting our two objectives: the reputation model, and the free-movement model.

4.1 Experimental Design

It is important for an experimental design to help determine whether a new heuristic method contributes something important. Barr et al. [9] present a list of possibilities. A heuristic method makes a contribution if it is:

- Fast: produces high-quality solutions quicker than other approaches;
- Accurate: identifies higher-quality solutions than other approaches;
- Robust: less sensitive to differences in problem characteristics, data quality, and tuning parameters than other approaches;
- Simple: easy to implement;
- High-impact: solves a new or important problem faster and more accurately than other approaches;
- Generalizeable: having application to a broad range of problems;
- Innovative: new and creative in its own right.

Barr furthermore asserts [9] that research reports about heuristics are valuable if they are:

- Revealing: offering insight into general heuristic design or the problem structure by establishing the reasons for an algorithm's performance and explaining its behavior;
- Theoretical: providing theoretical insights, such as bounds on solution quality

From Section 1.3 the *goal* of this research is to develop a scalable software architecture for a multi-agent, flow-based intrusion detection system. We have two high-level objectives:

- Design and evaluate a multi-agent intrusion detection system using a Reputation system
- Design and evaluate a multi-agent intrusion detection system using stochastic search

The heuristics defined by Barr are contained directly within the objectives: Reputation system and Evolutionary Algorithm. Each objective leads to a distinct experiment that must be conducted, and are defined in the following sections.

4.2 MFIRE-2 Reputation System Experimental Design

Testing the effectiveness of the MFIRE-2 MAS using Reputation consists of training two classifiers (local and combined) using data generated offline, and testing the MAS online in two modes of operation. In particular, we seek to find the overall MAS classification accuracy with a 4-agent and 8-agent model. For both models, we observe the accuracy at the initial time in the simulation, and at the end. The

change in accuracy during this time represents the increase in performance attained from agents finding better vantage points in the network.

To create the training data, we run repeated simulations on a single, 100-node regional network, where each simulation represents a single attack scenario. The attack is selected one-at-a-time from our four defined scenarios: Normal, DDoS, Worm, and Scan. Flow-based statistics are captured in two places in the network and processed to create two training sample sets, local data and combined data. We then train both classifiers with an SVM using an RBF Kernel, 5-fold cross validation, and a grid search for optimal parameters C and γ. The two resulting classifier models are used for all subsequent testing.

To limit the variance in the experiment, all four scenarios are executed 20 times, and average accuracy recorded. We perform 30 sample observations, which yields 4800 total simulations needed. This sample size is chosen to allow first and second order statistics to be used to evaluate the results. More observations are preferred, and 30 is an acceptable number to achieve a good confidence level while still running in a reasonable time. Each simulation must be performed for a minimum number of necessary time steps to ensure agents have time to move to better vantage points, and a time span of 80 time periods is conservatively allocated.

Experiment summary for the Reputation model:

- Four attack scenarios: Normal, DDoS, Worm, Scan
- Each simulation: 80 time periods
- One observation: four scenarios over 20 simulations each
- Total sample size: 30 observations
- Number of agents: four or eight

4.3 MFIRE-2 Evolutionary Algorithm Experimental Design

This section defines the approach used to qualitatively validate the functionality of the free-movement model, using a genetic algorithm. The free-movement model is created by running a generational genetic algorithm on a population of candidate solutions. Each successive generation, the individuals become more adapted to solving the problem. We conduct the GA with the following parameters to qualitatively validate its functionality:

- Single objective: maximize overall classification accuracy
- Individual solution: real-valued agent movement actuator
- Feature selection: two random features selected
- Maximum evaluations: three generations

The fitness function for this experiment is defined as:

- Four agents in the network
- Four scenarios: Normal, DDoS, Scan and Worm
- Simulated for two observation periods
- Improvement: difference between final accuracy and initial accuracy
- Fitness: average improvement over 2 simulations

The test parameters and fitness function defined here are not meant as a complete method to find a near-optimal movement actuator, but rather as a method for validating the core functionality of the MFIRE-2 free-movement model. Specific qualitative test cases for the free-movement model are:

- Fitness function
- Polynomial mutation
- Simulated Binary Crossover (SBX)
- Binary tournament selection
- Parent-child replacement
- Convert actuator to chromosome
- Convert chromosome to actuator
- Save actuator model
- Read actuator model
- Agent behavior

The final results of both the Reputation model and free-movement model are presented in the next section.

4.4 Analysis

The next two sections provide the detailed results for the experiments defined in Section 4.1.

4.4.1 MFIRE-2 Reputation System Performance Assessment.

The MFIRE-2 system is tested with the plan described in Section 4.2. Figure 13 shows the number of agents that move in each time period. Figures 14 and 15 show the average accuracy and false positive rate for the MAS at every time period, for

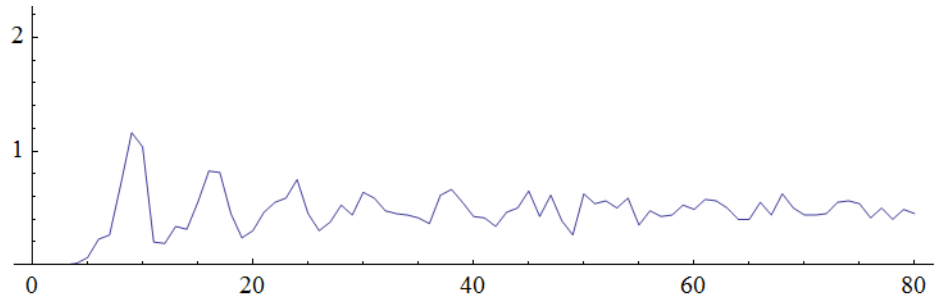


Figure 13. Average number of agents moving each period

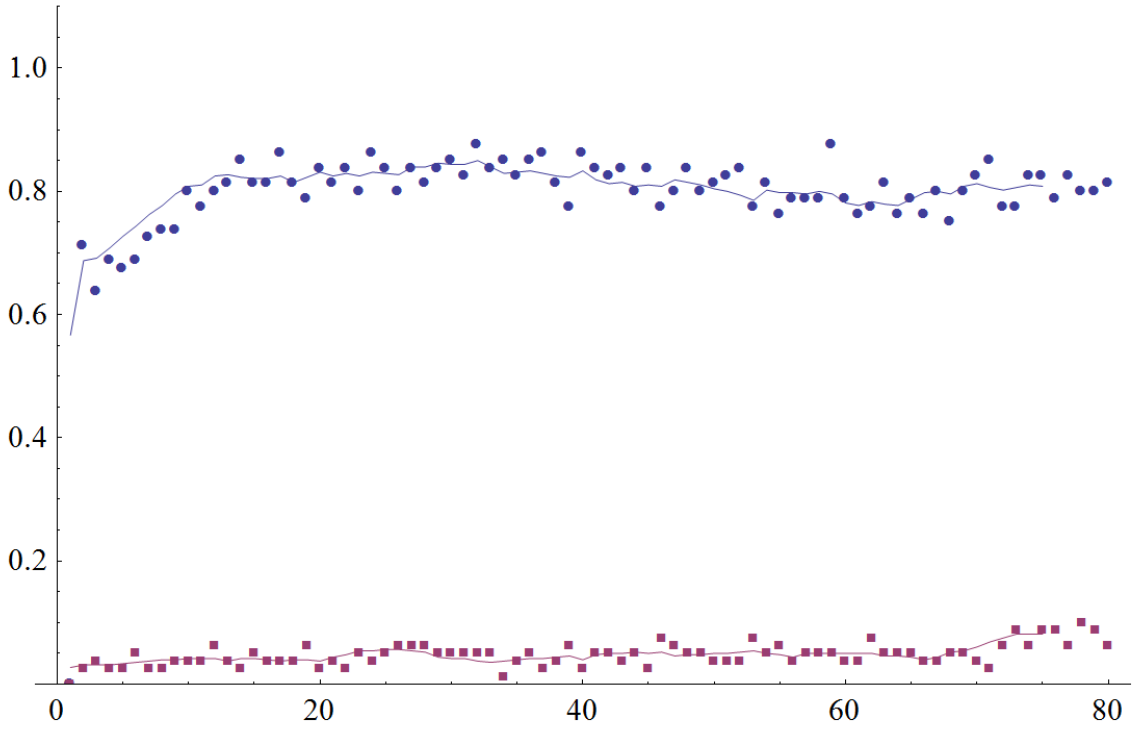


Figure 14. 4 agents using reputation model: accuracy (upper curve) and false positives (lower curve) vs. time

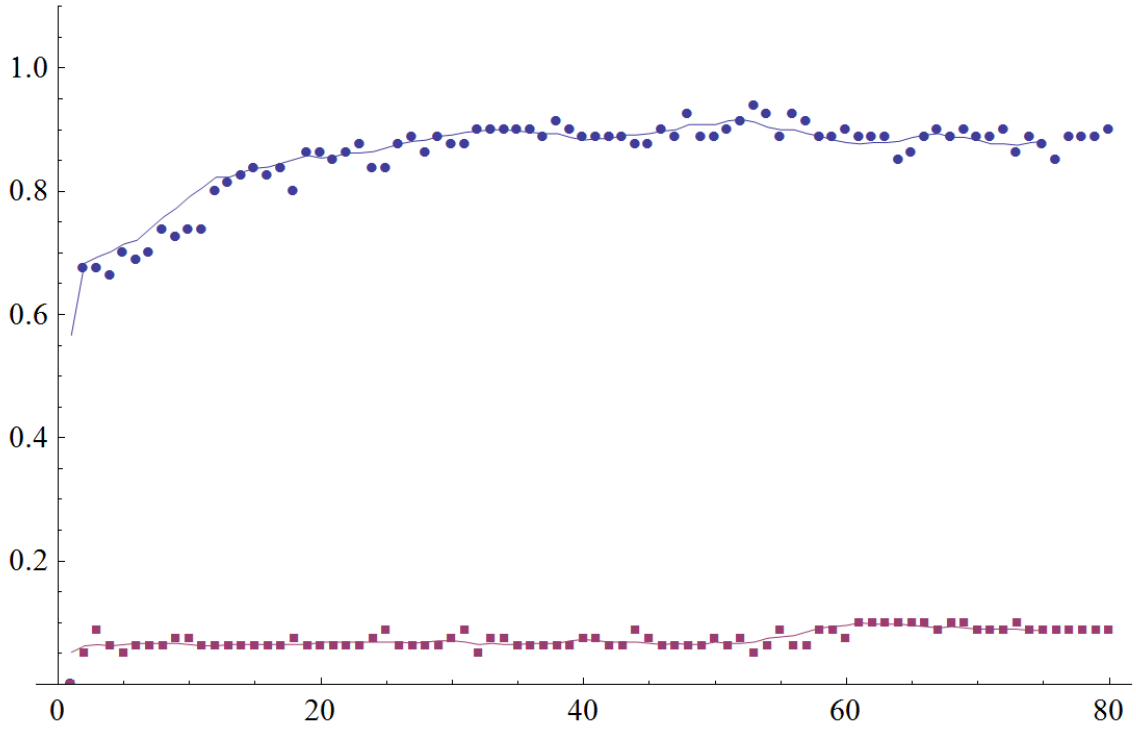


Figure 15. 8 agents using reputation model: accuracy (upper curve) and false positives (lower curve) vs. time

the 4-agent and 8-agent models. Each data point is the average of 80 simulations: 20 for each of the four scenarios.

We now evaluate the relative performance of Reputation system when using 4 or 8 agents in network. When making a classification, the result can be *correct*, a *false positive*, or a *false negative* (see Section 2.2). $Accuracy = \frac{numbercorrect}{total}$. Accuracy data for all 30 sample observations is provided in Table 5. The columns show results for each of the two experiments: 4-agent and 8-agent, and provides both the initial and final accuracy. The same data is shown as a histogram in Figures 16 and 17, and as a box plot in Figure 18.

To compare any two of the models we use Wilcoxon Rank Sum (or Mann-Whitney) test, which is a non-parametric statistical hypothesis test for assessing whether one of two samples of independent observations tends to have larger values than the other.

Table 4. Reputation System Overall Accuracy

	Initial: 4-agent	**Initial: 8-agent**	**Final: 4-agent**	**Final: 8-agent**
mean	0.678	0.638	0.821	0.890
median	0.688	0.650	0.820	0.906
stddev	0.038	0.043	0.032	0.043

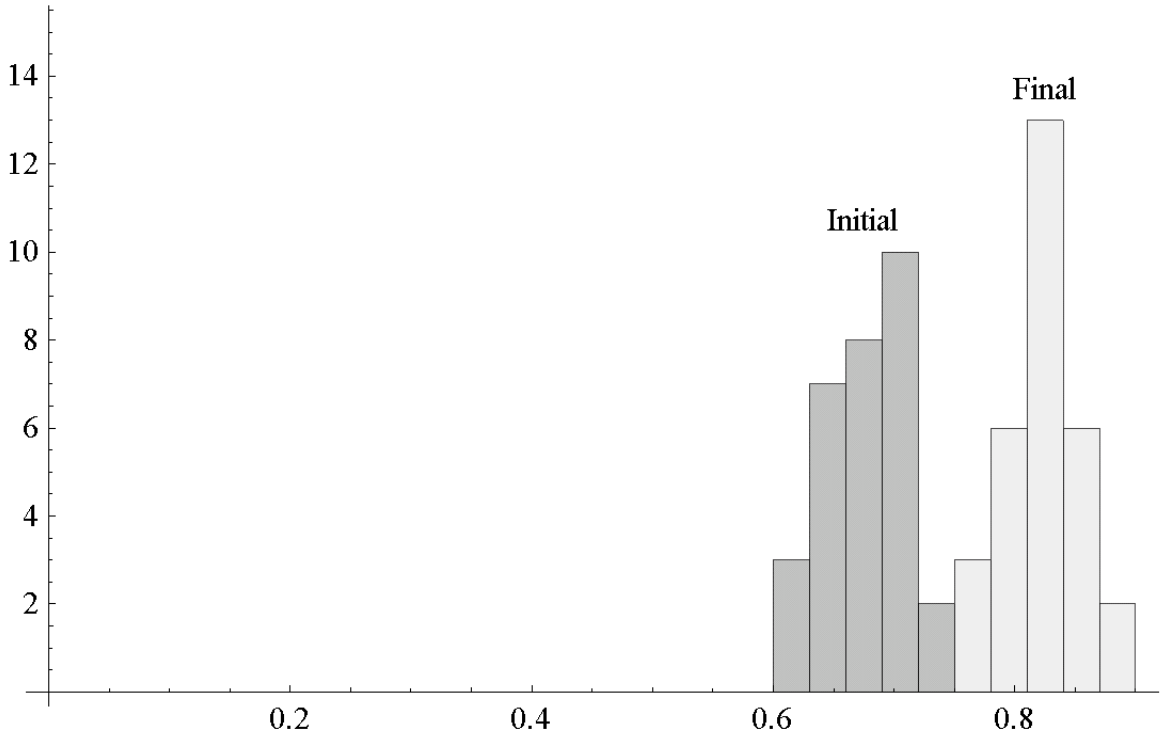


Figure 16. Accuracy histogram: 4 agents

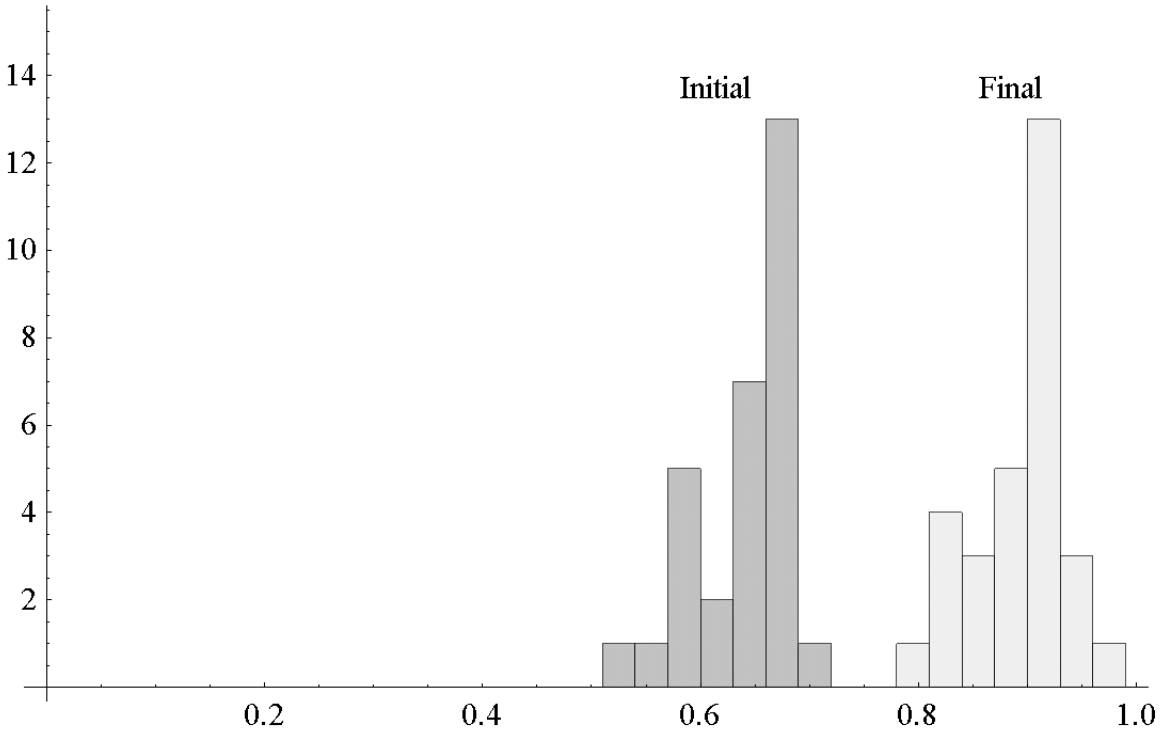


Figure 17. Accuracy histogram: 8 agents

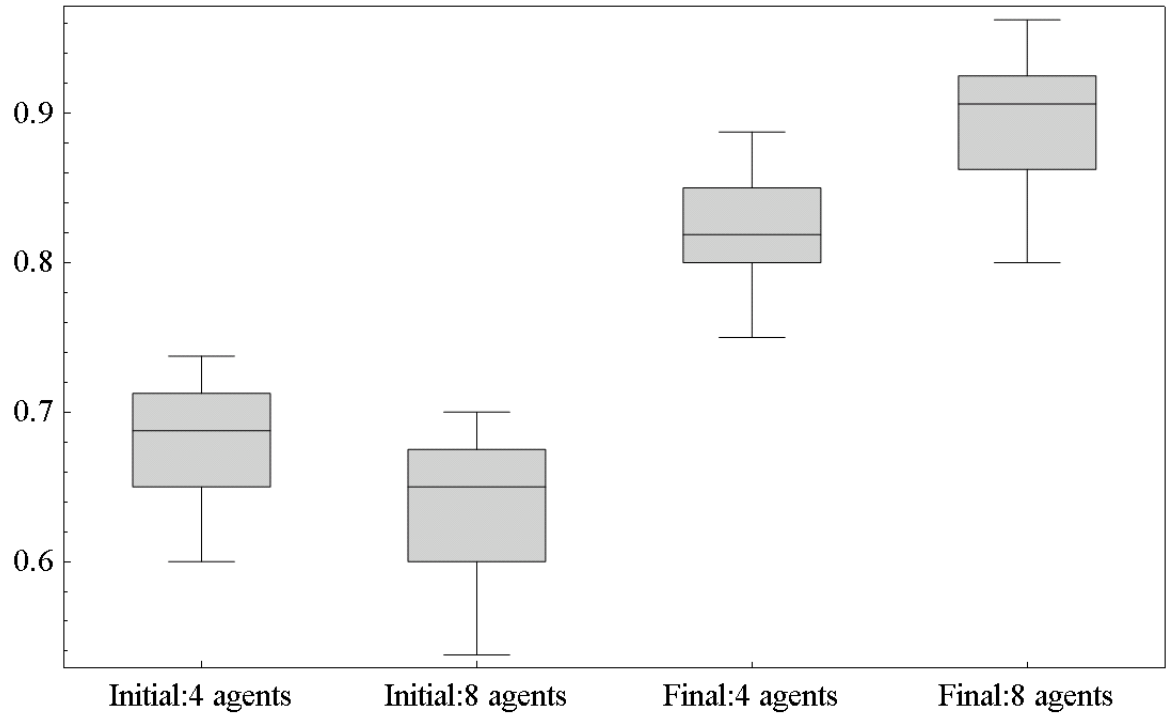


Figure 18. Accuracy box plots

Table 5. Wilcoxon Rank Sum p-values

	p-value
Initial:4-agent vs. Final:4-agent	2.1×10^{-22}
Initial:8-agent vs. Final:8-agent	1.8×10^{-30}
Initial:4-agent vs. Initial:8-agent	4.5×10^{-4}
Final:4-agent vs. Final:8-agent	3.3×10^{-9}

It remains the logical choice when the data are ordinal but not interval scaled, so that the spacing between adjacent values cannot be assumed to be constant. The MannWhitney test is more robust than the Student t-test, as it is less likely to spuriously indicate significance because of the presence of outliers. We use the MATLAB *ranksum* function to compare of all six possible combinations of the three models under test. A p-value of less than 0.05 indicates a significant difference between the two models under comparison, with 99% confidence. A p-value of larger than 0.05 indicates there is not sufficient evidence that the two models perform differently.

4.4.2 MFIRE-2 Evolutionary Algorithm Performance Assessment.

The free-movement model is successfully validated by performing the following individual tests. Observations are performed during run-time code debugging, as well as a correctly formed outputs.

- Fitness function—Correctly calculates the average accuracy improvement over 4 scenarios and 2 simulations each
- Polynomial mutation—jMetal executes all mutation operations successfully, and new actuator is created with the correct new real-valued parameters
- SBX crossover—jMetal executes all crossover operations successfully, and two new child actuators are created with the correct new real-valued parameters, from two given parent actuators

- Binary tournament selection—jMetal correctly selects parents based on fitness
- Parent-child replacement—Children replace their own parents during crossover operations
- Convert actuator to chromosome—The movement actuator (n-dimensional class structure) is converted to a jMetal decision variable (one-dimensional, real-valued list)
- Convert chromosome to actuator—The jMetal decision variable (one-dimensional, real-valued list) is converted to a movement actuator (n-dimensional class structure)
- Save actuator model—Movement actuator model is saved to a new text file
- Read actuator model—Movement actuator is read from an existing text file model
- Agent behavior—All agents in the MAS operate in accordance with their local movement actuator. The Agent Controller no longer dictates agent movement. The voting system, reputation system, SVM classifier, Agent Managers, and all networking features operate correctly while in this mode

This concludes the testing and validation of the two MFIRE-2 classification models. The complete quantitative analysis is performed for the Reputation model, using a 4-agent and 8-agent system. The results gained from this experiment provide a well-formed baseline to compare with potential future approaches. An additional qualitative analysis of the free-movement model indicates that all features function correctly, and it is ready for additional testing. The final conclusions for this research are provided in the next chapter, as well as some suggestions for future work.

V. Conclusions and Future Research

The *goal* of this research is to continue development of a scalable software architecture for a multi-agent, flow-based intrusion detection system. The following high-level *objectives* support this goal:

- Design and evaluate a multi-agent intrusion detection system using a Reputation system
- Design and evaluate a multi-agent intrusion detection system using stochastic search

We are successful in achieving both objectives. The original MFIRE 1.0 framework is updated to indentify simulated network attacks using a SVM classifier. In addition, two models of agent movement are introduced: one based on a central reputation system, the other optimized using a genetic algorithm. For the first objective, A baseline experiment is conducted to find the performance of the MAS when agents do not move (68% accuracy) (see Section 4.4.1). This represents the lower bound of classification performance. Moving agents for both the 4-agent model (82% accuracy) and 8-agent model (89% accuracy) provide a conclusive increase in classification performance over fixed agents. In both cases, the reputation model's lowest observed final accuracy is higher than the model's highest observed initial accuracy. This clearly indicates greater performance when agents can find better vantage points, and is supported by the low p-values with the Wilcoxon Rank Sum test.

One may ask why the attained accuracy is not closer to 100%. The reputation model and free-movement model introduced in this research are not a panacea. In general, there are five sources of error that prevent the MAS from achieving 100% accuracy:

1. Limited flow-based statistics are collected on every node
2. The quality/amount of data used for training
3. The ability of the classifier training algorithm to construct a good model
4. The voting system used to convert individual agent classifications into a final combined classification
5. Number and location of agents in the network

Essentially, our research focuses on holding the first four items constant, while performing experiments on number 5. Clearly there may be ways to improve the other aspects listed above, and should be explored in future research. However, the ultimate goal of developing a scalable software architecture for intrusion detection is realized. MFIRE-2 has shown good comprehensive performance in defending military networks from attacks, and should be carefully considered for implementation in cyber warfare applications.

If selected for continued research or implementation, one key experiment that should be conducted is to find the maximum potential classification performance of the system. This research describes the "lower bound" of performance as a set of non-moving agents. However, we have yet to test any "upper bound". Indeed this is harder to quantify, but one possible approach may be to simply increase the number of agents to the maximum. That is, set one fixed agent at every node. This essentially takes movement out of the equation. Reputation could still be maintained for feature sharing assignments. However, no assessment is done to validate how the reputation and voting system would respond in this situation. In particular, agents would no longer have their reputations reset by moving, which could have unexpected consequences.

These results open many additional avenues for future research. In particular, one of our original goals is to incorporate SOMAS work into MFIRE. SOMAS uses agents with many actuators to take action within the network. These agents are optimized using a genetic algorithm. It is possible that many different types of agents could exist on the network simultaneously, each providing specific functionality at specific locations. This thesis began this endeavor by creating a single *movement* actuator, but there is still much to do on this front. In particular, to run any stochastic search (e.g., genetic algorithm), MFIRE needs to be made more efficient in conducting simulations. Two approaches are needed to achieve this. First, code must be optimized to run simulations faster. This requires rethinking the way MASON is used to set up and run simulations, and the way agents move in the network. Up to this point, the focus was on well-formed code; not necessarily on efficiency. Second, simulations must be made to cleanly run in parallel. Fitness functions for a genetic algorithm would gain considerable speed-up from concurrent processing on a cluster.

A key element of MFIRE-2 is an inherent ability to detect changing attacks in a dynamic networking environment. Because the agents never actually converge to a perceived optimal vantage point, they continue to explore the network looking for new threats. This approach lends itself particularly well to a wireless environment, where network layout is less constant than a wired network. The focus of our research is on a single 100-node AS-level layout, but is certainly not limited to that. An exciting alternative would be to explore the system's capabilities in a generalized airborne wireless domain.

Other areas of future work are presented throughout the text, and we provide some additional items here. First, the attack models that are introduced are simplified versions of those seen in the real world. More complex, realistic models are possible, especially for worm attacks and vulnerability scans. Botnets provide an

exciting opportunity to combine the effects of multiple attacks into a single package. Ed Skoudis' book [80] provides a good summary of possible attacks. Second, the reputation and voting system could be improved by changing to a probabilistic model. Currently, agents in the system provide a single best-guess, which is combined into a simple majority vote. Another approach would be for agents to provide a probability estimate for each possible attack. Third, some knowledge of node reputation should be retained. As is, agents move to a new random node when their reputation drops low enough. But the low reputation they garnered at that location is lost, so in future time periods it is likely that another agent will move to this "bad" location. Keeping track of reputations at the node level could improve performance. Fourth, MFIRE may be expanded to simulate networks down to a finer level of detail than top-level autonomous systems and gateway routers. Although one may not want to simulate every routing detail in these sub networks, there is potential benefit in allowing agents to see explicit details of the source and destination addresses. Fifth, agents may be given access to information contained in packet headers. This provides a major advantage in detecting certain types of attacks, in particular vulnerability scans. Finally, while the framework introduced in this research is based around a simulated network environment, there is no perfect substitute for real-world network traffic. Future work should look at incorporating captured packet traffic on existing networks.

MFIRE is mature enough that these improvements and others can be incorporated readily. The object-oriented framework can be expanded or focused as needed to run many conceivable experiments with multiagent systems and intrusion detection.

Appendix A. MFIRE System Details

This appendix provides some additional MFIRE details for agent communication, reprinted from [37] for clarity. In addition, we include a detailed diagram depicting the relationship between online and offline modes of execution for MFIRE.

The figures provided in this section show the messages used in MFIRE. In each figure, the left side is used for the sender. The type of the message is displayed first, and below it, the format. The format is essential for extracting message components from the packet's payload, which itself is a single string. On the right side of each figure, we show the actions that are taken by the recipient.

Figure 19 shows the messages sent from the controller and received by agents.

Figure 20 shows the messages sent from agents and received by the controller.

Figure 21 shows the SHARE message used for feature value exchange between agents.

Figure 22 shows the messages involved in agent migration. MIGRATE is sent by an agent that received MOVE from the controller previously. It is sent to the AgentManager at the migration destination node. The MIGRATE message contains all information required to reinstantiate the agent at the distant end. MIGRATEACK is sent by an AgentManager that received a MIGRATE message previously. It is sent to the AgentManager at the node where the original copy of the migrating agent still resides. The AgentManager that receives MIGRATEACK terminates the agent.

Figure 23 depicts the relationship between offline training and online testing of a classifier, when using a SVM trained with libsvm. The bottom half lays out the execution of initial offline training, which generates a SVM classifier model and associated scaling file. The top half represents the online testing of a classifier. It uses the existing classifier model and scale file.

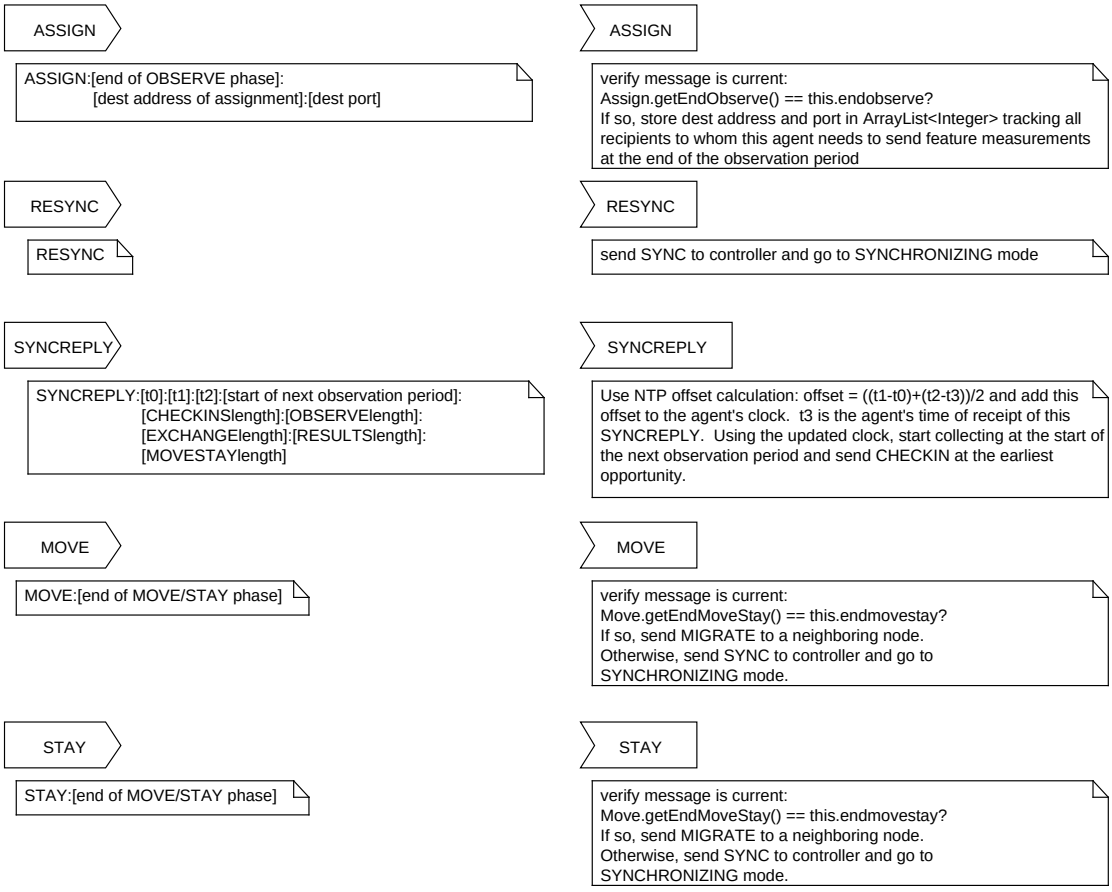


Figure 19. MFIRE: Messages sent by the controller and received by agents

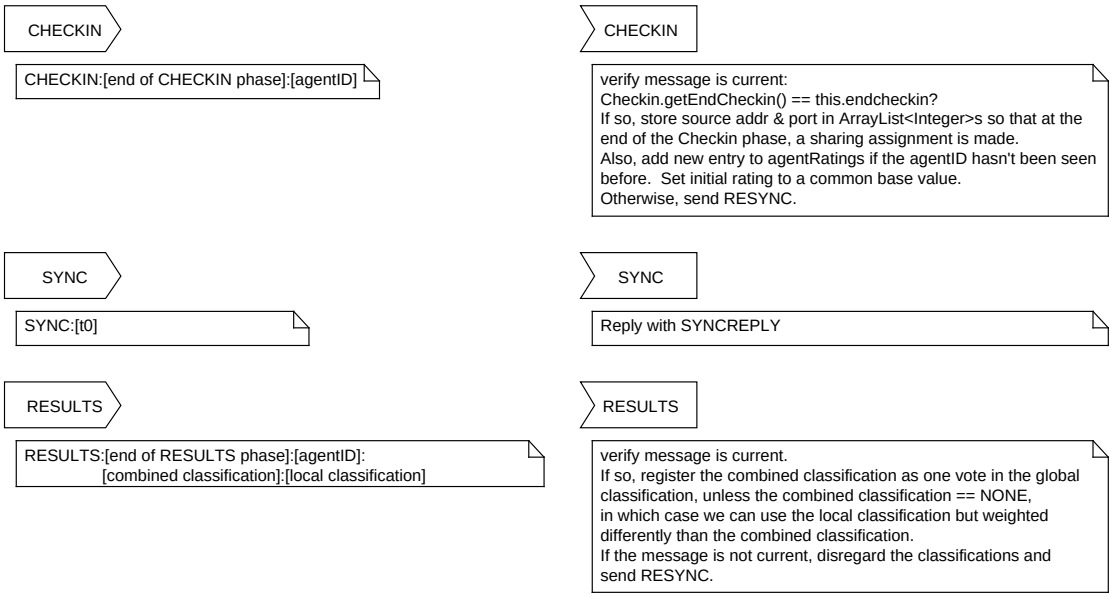


Figure 20. MFIRE: Messages sent by agents and received by the controller

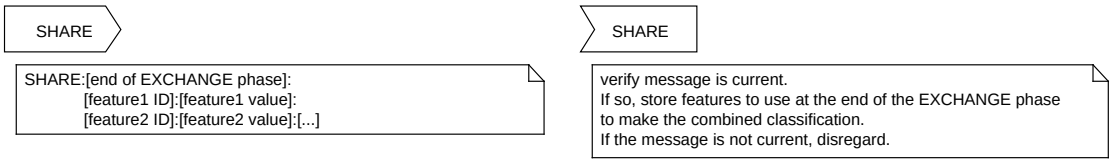


Figure 21. MFIRE: Messages sent by agents to other agents

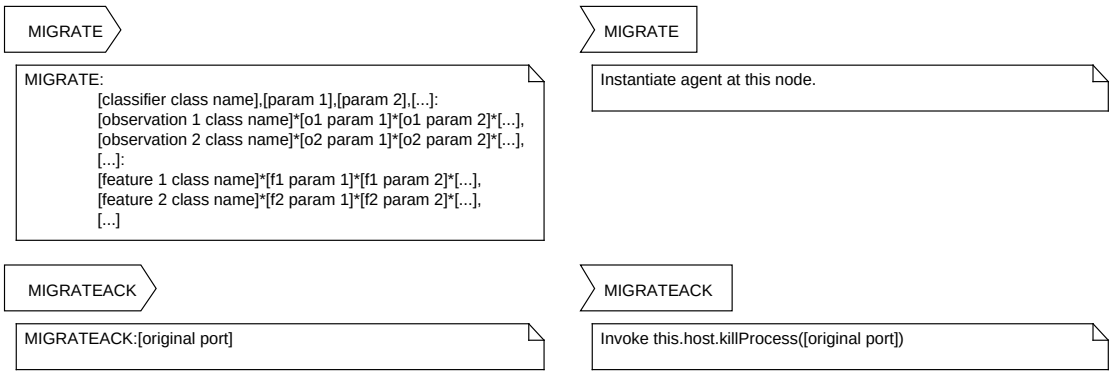


Figure 22. MFIRE: Messages involved in agent migration

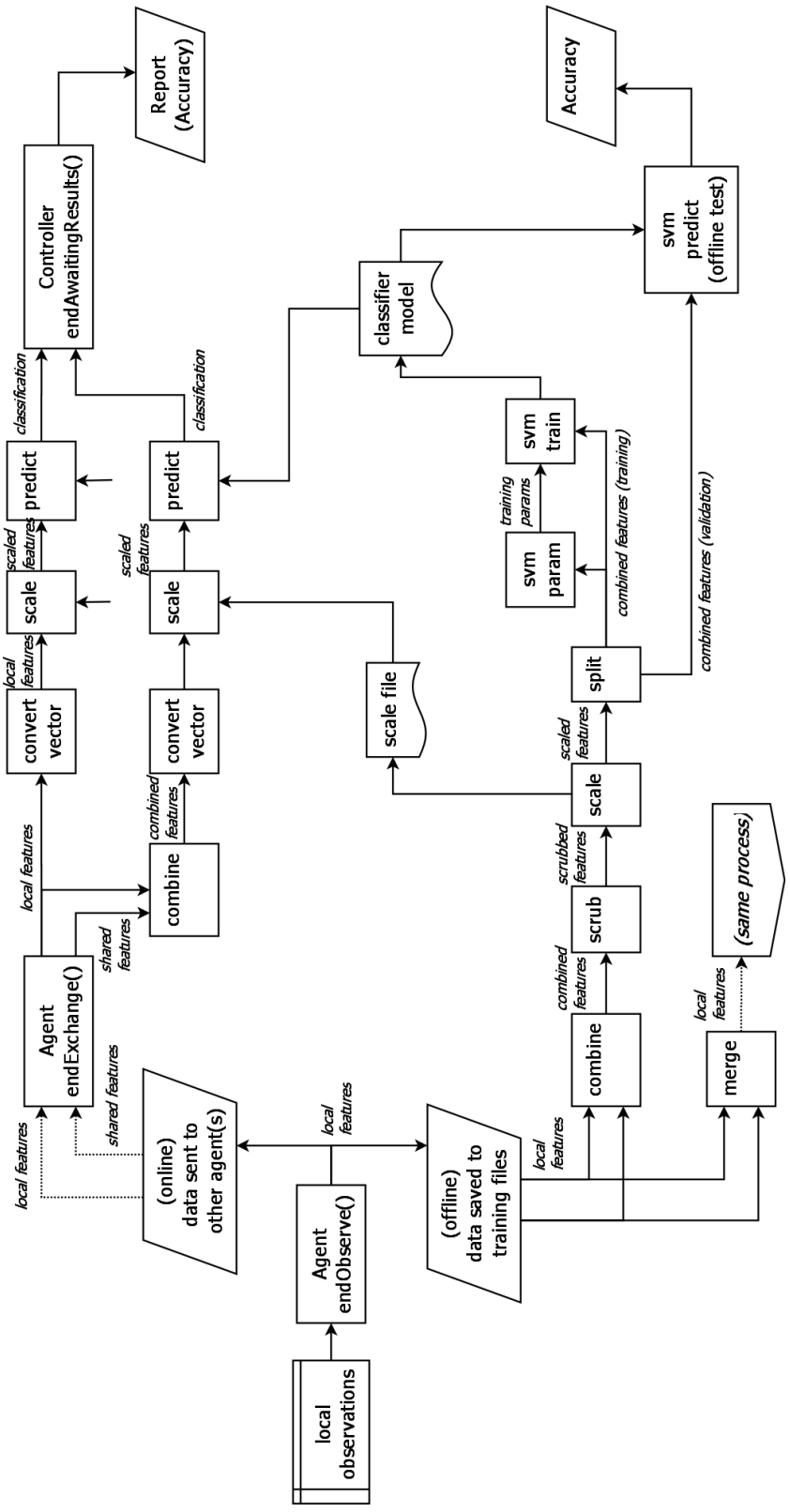


Figure 23. MFIRE detailed offline training and online testing execution paths

Appendix B. MFIRE Change Log

This section annotates the changes made from version 1.0 of MFIRE.

- New class `ScenarioNetwork`: Takes network and process parameters from an input file instead of defined in code. This helps guarantee experiments are consistent when run in online and offline modes (data generation versus testing). This class replaces the need for the other individual network representations (but not preclude thier use if needed).
- New class `DataGenerator`: Based on the `Experiments` class, `DataGenerator` runs a simulation to generate training data.
- New class `SVMScale`: Used by classifiers to consistently scale data. Reads scale files in libsvm format, and creates scale files during data generation.
- New class `OnlineTest`: Code base for running online simulations.
- Added new constructor to several `mfire.process` classes which allows them to take a string of arguments during creation. This is used by `ScenarioNetwork`.
- Bug fix: Observation `O_DestFlows` was updated to change an internal variable (hash set `destflows`) from a class member to a local, like all the other observations classes.
- New class `SVMClassifier`: uses libsvm to perform classification.
- `AgentController` now records classification output to a file, during online testing.
- Bug fix: `Agent` class now uses a different delimiter for migrate messages. The original caused crashes during moves.

- New class `OfflineTest`: uses existing simulation data to test classifier accuracy. This was mostly used for debugging, and is not strictly needed to perform the experiments listed in this thesis.
- Bug fix: `MF_Node.ProcessTopgenNodeLine` now uses scale instead of TTL when creating a new node.
- Fixed: Agent IDs are now being maintained between moves.
- Fixed: Reputation updates are now being correctly applied to the provider of shared features, instead of the recipient.
- Bug fix: Agent reputation is now being reset after a move.
- Reputation change: when resetting reputation, a small plus-or-minus offset is added to the default value. This fixes a problem where, rarely, agents would move in lock step with each other and never explore the network in the preferred stochastic manner.
- New class `Actuator`: serves as the base class for all agent actuators, similar to `Classifier`. They take a feature set as input and output a binary "fire" decision.
- New class `Segmented2DActuator`: one instance of an actuator. Takes two features, and segments the firing decision into a probabilistic activation map. Chosen for its ease of manipulation during genetic operations (mutation and crossover).
- Integration with jMetal: Now calls the jMetal framework to run a single-objective generational genetic algorithm. Uses a `Segmented2DActuator` for decision variables. Implemented as a custom replacement for the jMetal `gGA` class.

- New class `SimulatedAnnealing`: implements a simulated annealing algorithm, as an alternative to a GA.

Appendix C. Evolutionary Algorithms: Details and Applications

This appendix provides formalisms for generic, single objective evolutionary algorithms as well as some examples of how evolutionary algorithms have been applied.

3.1 Evolutionary Algorithms: Details

For a single objective evolutionary algorithm, Bäck defines an EA as an 8-tuple[4]:

$$\text{EA} = (I, \Phi, \Omega, \Psi, s, \iota, \mu, \lambda)$$

where I is the space of individuals (analagous to the feasible region M). $\Phi : I \to \mathbb{R}$ is a fitness function that assigns real values to individuals based on performance with respect to the *objective*.

$$\Omega = \{\omega_{\Theta_1}, \ldots, \omega_{\Theta_z} | \omega_{\Theta_i} : I^\lambda \to I^\lambda\} \cup \{\omega_{\Theta_0} : I^\mu \to I^\lambda\}$$

is a set of probabilistic genetic operators ω_{Θ_i} such as mutation or recombination. Each operator is controlled by specific parameters summarized in the sets $\Theta_i \subset \mathbb{R}$.

The *selection operator* is

$$s_{\Theta_s} : (I^\lambda \cup I^{\mu+\lambda}) \to I^\mu$$

which may change the number of individuals from λ or $\lambda + \mu$ to μ, where $\mu, \lambda \in \mathbb{N}$ and $\mu = \lambda$ is permitted. Here, λ is the number of offspring and μ is the number of parents in the population. As with the probabilistic genetic operators, the selection operator may make use of a set of parameters; specifically, Θ_s.

The *termination criterion* for the EA is expressed by $\iota : I^\mu \to \{\text{true}, \text{false}\}$. The

generation transition function $\Psi : I^{\mu} \rightarrow I^{\mu}$ specifies the complete process by which we transition from the population P of the current generation to that of the subsequent generation:

$$\Psi = s \circ \omega_{\Theta_{i_1}} \circ \ldots \circ \omega_{\Theta_{i_j}} \circ \omega_{\Theta_{i_0}}$$
$$\Psi(P) = s_{\Theta_s}(Q \cup \omega_{\Theta_{i_1}}(\ldots(\omega_{\Theta_{i_j}}(\omega_{\Theta_{i_0}}(P)))\ldots))$$

Here, $\{i_1, \ldots, i_j\} \subseteq \{1, ..., z\}$. That is to say, Ψ is permitted to use a subset of available, parameterized operators. Finally, $Q \in \{\emptyset, P\}$. This way, Ψ may specify whether selection includes the population as it existed prior to the current generation's transformation along with the transformed population. The operator $\omega_0 : I^{\mu} \rightarrow I^{\lambda}$ serves to change the population size so that the required λ offspring individuals are produced from the μ parents. Selection reverts the population size to μ.

A *population sequence* $P(t+1) = \Psi(P(t)), \forall t \geq 0$ naturally results from Ψ, where t denotes the generation. Members of $P(0)$ are typically initialized randomly, but $P(0)$ may also be generated from a specified starting point.

Genetic operators come in many varieties but are broadly categorized by the number of "parents" involved. Mutation is an example of an asexual genetic operator, while recombination is typically sexual, involving two parents, but could be extended (without basis in biology) to involve arbitrarily many parents.

Permissible mutations depend on the representation of the individual. A common case is the bitstring representation. A bit-flip mutation flips the selected bit with some probability. Similar single-parameter mutations for parameters in other domains may involve an attempt to constrain the mutation so that the new value is "close" by some measure to the old value (for example via shifting by a value drawn from a gaussian distribution).

Other genetic operators are similarly representation-dependent. Common recombination operators include one-point crossover, where a parameter position is selected at random via some distribution (e.g. uniform), and the offspring is formed by copying the all parameter values in the first parent up to and including the crossover point, and taking the rest of its values from the second parent after the crossover point. In general, n-point crossover extends this concept to n randomly selected crossover points.

Algorithm 3 Outline of an Evolutionary Algorithm

$t := 0$
initialize $P(0) := \{\vec{a}_1(0), \ldots, \vec{a}_\mu(0)\} \in I^\mu$
evaluate $P(0) : \{\Phi(\vec{a}_1(0)), \ldots, \Phi(\vec{a}_\mu(0))\}$
while $\iota(P(t)) \neq$ **true do**
  recombine: $P'(t) := r_{\Theta_r}(P(t))$
  mutate: $P''(t) := m_{\Theta_m}(P'(t))$
  evaluate: $P''(t) : \{\Phi(\vec{a}''_1(t)), \ldots, \Phi(\vec{a}''_\lambda(t))\}$
  select: $P(t+1) := s_{\Theta_s}(P''(t) \cup Q)$
  $t := t + 1$
end while

The general outline of an Evolutionary Algorithm is presented by Bäck in algorithm 3 [4].

3.2 Evolutionary Algorithms: Applications

A particularly pertinent example of the application of Evolutionary Algorithms to pattern recognition comes from Radtke et al. [72]. The authors apply Multi-Objective Genetic Algorithms (MOGAs) to two parts of a handwritten character recognition system. First, they use the Multi Objective Memetic Algorithm (MOMA) [73] to extract a small set of effective features. The fitness function minimizes both the dimensionality of the feature set and the classification error rate of a projection distance (PD) classifier [50] on a validation set.

The PD is only used as a wrapper for evaluation of the extracted feature set. In the next step, these features are used to train a diverse set K of p classifiers to be used as candidates for the final Ensemble of Classifiers [51]. Again, a MOGA is applied, in this case the popular Non-Dominated Sorting Genetic Algorithm (NSGA-II) [22], for the subset selection from K. The fitness of each solution (a binary vector of length p indicating membership of each K_i in the candidate ensemble) in this case is based on the performance of the ensemble on a validation set.

Another pattern recognition example is presented in [98], in which the authors study feature selection using single and multi-objective memetic frameworks.

For examples of how these concepts have been applied to intrusion detection, see [5] for an analysis of MOGAs used to identify encrypted traffic. In [34], Haag applies a multi-objective artificial immune system to some public network traffic data sets to detect intrusions.

Bibliography

[1] "Snort". URL `www.snort.org`.

[2] "Department of Defense Strategy for Operating in Cyberspace", July 2011.

[3] "Port Numbers", February 2011. URL `http://www.iana.org/assignments/port-numbers`.

[4] Bäck, T. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA, 1996.

[5] Bacquet, C., A. Zincir-Heywood, and M. Heywood. "An Investigation of Multi-objective Genetic Algorithms for Encrypted Traffic Identification". *Computational Intelligence in Security for Information Systems*, 93–100, 2009.

[6] Bagrodia, R., R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song. "Parsec: A parallel simulation environment for complex systems". *Computer*, 31(10):77–85, 2002. ISSN 0018-9162.

[7] Banks, J., B.L. Nelson, and D.M. Nicol. *Discrete-event system simulation*. Prentice Hall, 2009. ISBN 0136062121.

[8] Bar, S., M. Gonen, and A. Wool. "A geographic directed preferential Internet topology model". *Computer Networks*, 51(14):4174–4188, 2007. ISSN 1389-1286.

[9] Barr, R.S., B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart. "Designing and reporting on computational experiments with heuristic methods". *Journal of Heuristics*, 1(1):9–32, 1995. ISSN 1381-1231.

[10] Bartos, Karel, Martin Grill, and Vojttech Krmicek. "Flow Based Network Intrusion Detection System using Hardware-Accelerated NetFlow Probes". *CESNET Conference 2008 Proceedings, pp. 49-56*. 2008.

[11] Brewin, Bob. "Defense Technology to Grow Despite Pentatgon Budget Cuts", January 2012. URL `www.nextgov.com`.

[12] Burgess, Mark. "CFEngine". URL `www.cfengine.com`.

[13] Burnett, Chris, Timothy J. Norman, and Katia Sycara. "Sources of Sterotypical Trust in Multi-Agent Systems". *Trust in Agent Societies, 14th Edition*. 2011.

[14] Calvert, K.I., M.B. Doar, and E.W. Zegura. "Modeling Internet topology". *Communications Magazine, IEEE*, 35(6):160 –163, June 1997. ISSN 0163-6804.

[15] Chang, Chih-Chung and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[16] Chapelle, O., P. Haffner, and V.N. Vapnik. "Support vector machines for histogram-based image classification". *Neural Networks, IEEE Transactions on*, 10(5):1055–1064, 2002. ISSN 1045-9227.

[17] Chen, Z., L. Gao, and K. Kwiat. "Modeling the spread of active worms". *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, 1890–1900. IEEE, 2003. ISBN 0780377524. ISSN 0743-166X.

[18] Cheng, L. *Simulation and topology generation for large-scale distributed systems*. Ph.D. thesis, UNIVERSITY OF BRITISH COLUMBIA, 2009.

[19] Cheng, L., N.C. Hutchinson, and M.R. Ito. "RealNet: A topology generator based on real internet topology". *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, 526–532. IEEE, 2008.

[20] Cid, Daniel. "OSSEC". URL `www.ossec.net`.

[21] Coello, C.A.C., G.B. Lamont, and D.A. Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer-Verlag New York Inc, 2nd edition, 2007.

[22] Deb, K., S. Agrawal, A. Pratap, and T. Meyarivan. "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II". *Parallel Problem Solving from Nature PPSN VI*, 849–858. Springer, 2000.

[23] Devroye, Luc. "Random variate generation in one line of code". *Proceedings of the 28th conference on Winter simulation*, WSC '96, 265–272. IEEE Computer Society, Washington, DC, USA, 1996. ISBN 0-7803-3383-7. URL `http://dx.doi.org/10.1145/256562.256623`.

[24] Doar, M.B. "A Better Model for Generating Test Networks". *Conference record*, 86. Institute of Electrical and Electronics Engineers, 1996.

[25] Erriquez, Elisabetta. "An abstract framework for reasoning about trust". *Trust in Agent Societies, 14th Edition*. 2011.

[26] Fabrikant, A., E. Koutsoupias, and C. Papadimitriou. "Heuristically optimized trade-offs: A new paradigm for power laws in the Internet". *Automata, Languages and Programming*, 781–781, 2002.

[27] Falliere, N., L.O. Murchu, and E. Chien. *W32. Stuxnet Dossier*. Technical report, Symantec, 2011.

[28] Faloutsos, M., P. Faloutsos, and C. Faloutsos. "On power-law relationships of the internet topology". *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 251–262. ACM, 1999. ISBN 1581131356.

[29] Floyd, R.W. "Algorithm 97: shortest path". *Communications of the ACM*, 5(6):345, 1962. ISSN 0001-0782.

[30] Fosnock, C. "Computer Worms: Past, Present, and Future". *East Carolina University*, 2005.

[31] Franklin, S. and A. Graesser. "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents". *Intelligent Agents III Agent Theories, Architectures, and Languages*, 21–35, 1997.

[32] Fujimoto, Richard M. "Parallel discrete event simulation". *Commun. ACM*, 33:30–53, October 1990. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/84537.84545`.

[33] Gordon, J. "Pareto process as a model of self-similar packet traffic". *Global Telecommunications Conference, 1995. GLOBECOM'95., IEEE*, volume 3, 2232–2236. IEEE, 2002. ISBN 0780325095.

[34] Haag, C.R., G.B. Lamont, P.D. Williams, and G.L. Peterson. "An artificial immune system-inspired multiobjective evolutionary algorithm with application to the detection of distributed computer network intrusions". *Proceedings of the 6th international conference on Artificial immune systems*, 420–435. Springer-Verlag, 2007.

[35] Hancock, D. and G. Lamont. "Multi Agent Systems on Military Networks". *IEEE Symposium on Computational Intelligence in Cyber Security*. 2011.

[36] Hancock, D. and G. Lamont. "Reputation in a multi agent system for flow-based network attack classification". *IEEE Symposium on Intelligent Agents*. 2011.

[37] Hancock, David. *A Multi-Agent System for Flow-Based Intrusion Detection Using Reputation and Evolutionary Computation*. Master's thesis, Air Force Institute of Technology, March 2011.

[38] Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer, 2nd ed. 2009. corr. 3rd printing edition, February 2009. ISBN 0387848576. URL `http://www-stat.stanford.edu/~tibs/ElemStatLearn/`.

[39] Hernandez-Pereira, E., J.A. Suarez-Romero, O. Fontenla-Romero, and A. Alonso-Betanzos. "Conversion methods for symbolic features: A comparison applied to an intrusion detection problem". *Expert Systems with Applications*, 36(7):612–617, 2009.

[40] Herrero, Alvaro and Emilio Corchado. "Multiagent Systems for Network Intrusion Detection: A Review". *Computational Intelligence in Security for Information Systems*, 63:143–154, 2009.

[41] Holloway, Eric. *Self Organized Multi Agent Swarms for Network Security Control*. Master's thesis, Air Force Institute of Technology, March 2009.

[42] Horng, Shi-Jinn, M. Su, and Y. Chen. "A novel intrusion detection system based on hierarchical clustering and support vector machines". *Expert Systems with Applications*, 1:306–313, 2011.

[43] Huynh, T.D., N.R. Jennings, and N.R. Shadbolt. "An integrated trust and reputation model for open multi-agent systems". *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006. ISSN 1387-2532.

[44] Ivanciuc, Ovidiu. "Applications of Support Vector Machines in Chemistry", 2007. URL `http://www.support-vector-machines.org/SVM_soft.html`.

[45] Jansen, W.A. "Intrusion detection with mobile agents". *Computer Communications*, 25(15):1392–1401, 2002. ISSN 0140-3664.

[46] Joachims, Thorsten. "SVM-light", August 2008. URL `http://svmlight.joachims.org/`.

[47] Josang, Audun, Roslan Ismail, and Colin Boyd. "A Survey of Trust and Reputation Systems for Online Service Provision". *Decision Support Systems*, 43:618–644, 2007.

[48] Karthick, R., Hattiwale V., and B. Ravindran. "Adaptive network intrusion detection system using a hybrid approach". *Fourth International Conference on Communication Systems and Networks (COMSNETS)*, 1:1–7, 2012.

[49] Kim, J., S. Radhakrishnan, and S.K. Dhall. "Measurement and analysis of worm propagation on Internet network topology". *Computer Communications and Networks, 2004. ICCCN 2004. Proceedings. 13th International Conference on*, 495–500. IEEE, 2005. ISBN 0780388143. ISSN 1095-2055.

[50] Kimura, F., S. Inoue, T. Wakabayashi, S. Tsuruoka, and Y. Miyake. "Handwritten numeral recognition using autoassociative neural networks". *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 1, 166–171. IEEE, 2002. ISBN 0818685123.

[51] Kittler, J., M. Hatef, R.P.W. Duin, and J. Matas. "On combining classifiers". *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):226–239, 2002. ISSN 0162-8828.

[52] Kong, J., M. Mirza, J. Shu, C. Yoedhana, M. Gerla, and S. Lu. "Random flow network modeling and simulations for DDoS attack mitigation". *Communications, 2003. ICC'03. IEEE International Conference on*, volume 1, 487–491. IEEE, 2003. ISBN 0780378024.

[53] Kotsiantis, S. B. "Supervised Machine Learning: A Review of Classification Techniques". *Informatica*, 31:249–268, 2007.

[54] Kurose, J. and K. Ross. *Computer Networking: A top-down approach*. Pearson Addison-Wesley, fifth edition, 2009.

[55] Liljenstam, M., Y. Yuan, BJ Premore, and D. Nicol. "A mixed abstraction level simulation model of large-scale Internet worm infestations". *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*, 109–116. IEEE, 2003. ISBN 0769518400. ISSN 1526-7539.

[56] Luke, Sean. "Multiagent Simulation and the MASON Library", August 2011. URL `http://cs.gmu.edu/eclab/projects/mason/`.

[57] Magoni, D. "nem: A software for network topology analysis and modeling". *10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, 364. 2002.

[58] McCanne, S., S. Floyd, and K. Fall. "ns2 (network simulator 2)". *last accessed: February*, 23, 2010.

[59] McDonald, C. "The cnet network simulator". *University of Western Australia*, 2003.

[60] McDonald, Chris. "The cnet network simulator". URL `http://www.csse.uwa.edu.au/cnet/index.html`.

[61] Medina, A., A. Lakhina, I. Matta, and J. Byers. "BRITE: An approach to universal topology generation". *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, 346–353. IEEE, 2002. ISBN 0769513158.

[62] Meier, J.D. "Improving Web Application Security: Threats and Countermeasures", June 2003. URL `http://msdn.microsoft.com/en-us/library/ff649874.aspx`.

[63] Ming-Yang and Su. "Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers". *Expert Systems with Applications*, 38(4):3492 – 3498, 2011. ISSN 0957-4174. URL `http://www.sciencedirect.com/science/article/pii/S0957417410009450`.

[64] Mirkovic, J. and P. Reiher. "A taxonomy of DDoS attack and DDoS defense mechanisms". *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004. ISSN 0146-4833.

[65] Moore, Andrew W., Denis Zuev, and Michael L. Crogan. *Discriminators for use in flow-based classification*. Technical report, Queen Mary University of London, 2005.

[66] Oetiker, Tobi. "RRDTool". URL `oss.oetiker.ch/rrdtool`.

[67] Park, Hyungwook and Paul A. Fishwick. "A GPU-Based Application Framework Supporting Fast Discrete-Event Simulation". *Simulation*, 86:613–628, October 2010. ISSN 0037-5497. URL `http://dx.doi.org/10.1177/0037549709340781`.

[68] Peng, Tao. *Defending Against Distributed Denial of Service Attacks*. Ph.D. thesis, The University of Melbourne, April 2004.

[69] Perdisci, Roberto, Davide Ariu, Prahlad Fogla, Giorgio Giacinto, and Wenke Lee. "McPAD : A Multiple Classifier System for Accurate Payload-based Anomaly Detection". *Computer Networks, Special Issue on Traffic Classification and Its Applications to Modern Networks*, 5:864–881, 2009.

[70] Postel, J. "RFC 793: Transmission control protocol", 1981.

[71] Postel, J.B. "User Datagram Protocol. RFC 768", 1980.

[72] Radtke, Paulo V. W., Robert Sabourin, and Tony Wong. "Classification system optimization with multi-objective genetic algorithms". Guy Lorette (editor), *Tenth International Workshop on Frontiers in Handwriting Recognition*. Université de Rennes 1, Suvisoft, La Baule (France), 10 2006. URL `http://hal.inria.fr/inria-00104200/en/`.

[73] Radtke, P.V.W., T. Wong, and R. Sabourin. "A multi-objective memetic algorithm for intelligent feature extraction". *Evolutionary Multi-Criterion Optimization*, 767–781. Springer, 2005.

[74] Resnick, P. and R. Zeckhauser. "Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system". *Advances in Applied Microeconomics: A Research Annual*, 11:127–157, 2002. ISSN 0278-0984.

[75] Russell, Stuart and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, December 2002. ISBN 0137903952.

[76] Scepanovic, Sanja. *Mitigating DDoS attacks with cluster-based filtering*. Master's thesis, Aalto University, June 2011.

[77] Sellke, S.H., N.B. Shroff, and S. Bagchi. "Modeling and automated containment of worms". *IEEE Transactions on Dependable and Secure Computing*, 71–86, 2007. ISSN 1545-5971.

[78] Shirey, R. "Internet Security Glossary, Version 2", August 2007. URL `http://tools.ietf.org/html/rfc4949`.

[79] Shoham, Y. and K. Leyton-Brown. *Multiagent systems: algorithmic, game-theoretic, and logical foundations*. Cambridge Univ Pr, 2008. ISBN 0521899435.

[80] Skoudis, Ed and Tom Liston. *Counter Hack Reloaded*. Prentice Hall, 2nd edition, January 2006.

[81] Soergel, D. "jlibsvm", 2009. URL `http://dev.davidsoergel.com/trac/jlibsvm`.

[82] Spatharis, A., I. Foudalis, M. Gjoka, P. Krouska, C. Amanatidis, C. Papadimitriou, and M. Sideri. "Improved tradeoff-based models of the Internet". *Proc. of SIWN/IEEE International Conference on Complex Open Distributed Systems*, volume 7. 2008.

[83] Specht, Ruby B., Stephen M.and Lee. "Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures". *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems, 2004 International Workshop on Security in Parallel and Distributed Systems*, 543–550. September 2004.

[84] Sperotto, A., G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. "An Overview of IP Flow-Based Intrusion Detection". *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010. ISSN 1553-877X.

[85] Talbi, E.G. *Metaheuristics: from design to implementation*. Wiley, 2009. ISBN 0470278587.

[86] Varga, A. et al. "The OMNeT++ discrete event simulation system". *Proceedings of the European Simulation Multiconference (ESM2001)*, 319–324. 2001.

[87] Wagner, A., T. D
"ubendorfer, B. Plattner, and R. Hiestand. "Experiences with worm propagation simulations". *Proceedings of the 2003 ACM workshop on Rapid Malcode*, 34–41. ACM, 2003. ISBN 1581137850.

[88] Wang, D., G. Chang, X. Feng, and Guo R. "Research on the Detection of Distributed Denial of Service Attacks Based on the Characteristics of IP Flow".

NPC Proceedings of the IFIP International Conference on Network and Parallel Computing, 1:86–93, 2008.

[89] Wang, Ke, Gabriela Cretu, and Salvatore Stolfo. *Anomalous Payload-based Worm Detection and Signature Generation*. Technical report, Columbia University, 2004.

[90] Weaver, Nicholas, Vern Paxson, Stuart Staniford, and Robert Cunningham. "A taxonomy of computer worms". *Proceedings of the 2003 ACM workshop on Rapid malcode*, WORM '03, 11–18. ACM, New York, NY, USA, 2003. ISBN 1-58113-785-0. URL `http://doi.acm.org/10.1145/948187.948190`.

[91] Whitman, Michael E. and Herbert J. Mattord. *Principles of Information Security*. Course Technology, 2011.

[92] Willinger, W., D. Alderson, and J. Doyle. "Mathematics and the Internet: a source of enormous confusion and great potential". *Notices of the American Mathematical Society*, 56(5):586–599, May 2009.

[93] Willinger, W. and V. Paxson. "Where mathematics meets the Internet". *Notices of the American Mathematical Society*, 45(8):961–971, 1998. ISSN 0002-9920.

[94] Winick, J. and S. Jamin. *Inet-3.0: Internet Topology Generator*. Technical Report UM-CSE-TR-456-02, Department of EECS, University of Michigan, 2002.

[95] Winter, P., E. Hermann, and M. Zeilinger. "Inductive Intrusion Detection in Flow-Based Network Data Using One-Class Support Vector Machines". *Proc. 4th IFIP Int New Technologies, Mobility and Security (NTMS) Conf*, 1–5. 2011.

[96] Zacharia, G. and P. Maes. "Trust management through reputation mechanisms". *Applied Artificial Intelligence*, 14(9):881–907, 2000. ISSN 0883-9514.

[97] Zeng, X., R. Bagrodia, and M. Gerla. "GloMoSim: a library for parallel simulation of large-scale wireless networks". *ACM SIGSIM Simulation Digest*, 28(1):154–161, 1998. ISSN 0163-6103.

[98] Zhu, Z., Y.S. Ong, and J.L. Kuo. "Feature Selection Using Single/Multi-Objective Memetic Frameworks". *Multi-Objective Memetic Algorithms*, 111–131, 2009.

[99] Zou, C.C., W. Gong, and D. Towsley. "Code red worm propagation modeling and analysis". *Proceedings of the 9th ACM conference on Computer and communications security*, 138–147. ACM, 2002. ISBN 1581136129.

[100] Zou, C.C., W. Gong, and D. Towsley. "Worm propagation modeling and analysis under dynamic quarantine defense". *Proceedings of the 2003 ACM workshop on Rapid Malcode*, 60. ACM, 2003. ISBN 1581137850.

Vita

Capt Timothy J. Wilson earned his Bachelor of Science degree in Computer Engineering at The University of Akron in August 2001. He was comissioned in the Air Force in February 2003 through Officer Training School (OTS), Maxwell Air Force Base, Alabama. From 2003-2006, Capt Wilson served as Foreign Materiel Exploitation Project Engineer for the National Air and Space Intelligence Center (NASIC), where he led teams to reverse engineer enemy electronic weapons (EW) systems. From 2006-2010, Capt Wilson served with the Electronic Systems Center (ESC) at Hanscom Air Force Base, Massachusetts. There he served on multiple programs including Airborne Maritime Fixed-Station Joint Tactical Radio System (AMF JTRS), Tactical Datalinks (TDL) Integration, and ESC/EN Executive Officer. Capt Wilson has earned certifications in the Acquisitions corps and Systems Engineering (SPRDE) career fields, and has been awarded the Air Force Commendation Medal. Upon graduation, Capt Wilson will be assigned to the 624th Operations Center at Lackland Air Force Base, San Antonio, Texas.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
22–03–2012	Master's Thesis	Sep 2010 — 22 Mar 2012

4. TITLE AND SUBTITLE

MFIRE-2: A Multi Agent System for Flow-Based Intrusion Detection Using Stochastic Search

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Wilson, Timothy J., Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCO/ENG/12-12

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally left blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT Detecting attacks targeted against military and commercial computer networks is a crucial element in the domain of cyber warfare. The traditional method of signature-based intrusion detection is a primary mechanism to alert administrators to malicious activity. However, signature-based methods are not capable of detecting new or novel attacks. This research continues development of a novel simulated, multiagent, flow-based intrusion detection system called MFIRE. Agents in the network are trained to recognize common attacks, and share data with other agents to improve the overall effectiveness of the system. A Support Vector Machine (SVM) is the primary classifier with which agents determine an attack is occurring. Agents are prompted to move to different locations within the network to find better vantage points, and two methods for achieving this are developed. One uses a centralized reputation-based model, and the other uses a decentralized model optimized with stochastic search. The latter is tested for basic functionality, and ready for continued experimentation. The reputation model is extensively tested in two configurations and results show that it is significantly superior to a system with non-moving agents.

15. SUBJECT TERMS

Flow Based, Intrusion Detection, Multi Agent Systems, Network Security

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gary B. Lamont (ENG)
U	U	U	UU	119	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4718; gary.lamont@afit.edu